SCONTENTS

Stores LaTeX contents

v2.2   2025-03-26[*]

©2019–2025 by Pablo González[†]

Abstract

This package allows to store LaTeX code, including "*verbatim*", in ⟨*sequences*⟩ using the l3seq module of expl3. The ⟨*stored content*⟩ can be used as many times as desired in the document, additionally you can write to ⟨*external files*⟩ or show it in ⟨*verbatim style*⟩.

Contents

💣 The next update removes compatibility with versions prior to 2024.

1 Description of the package

The SCONTENTS package allows to ⟨*store contents*⟩ in ⟨*sequences*⟩ or ⟨*external files*⟩. In some ways it is similar to the filecontentsdef package, with the difference in which the ⟨*content*⟩ is stored. The idea behind this package is to get an approach to ConTeXt "*buffers*" by making use ⟨*sequences*⟩.

2 Motivation and Acknowledgments

In LaTeX there is no direct way to record content for later use, although you can do this using \macros, recording ⟨*verbatim content*⟩ is a problem, usually you can avoid this by creating external files or boxes.

The general idea of this package is to try to imitate this implementation "*buffers*" that has ConTeXt which allows you to save content in memory, including *verbatim*, to be used later. The package filecontentsdef solves this problem and since expl3 has an excellent way to manage data, ideas were combined giving rise to this package.

This package would not be possible without the great work of JEAN FRANÇOIS BURNOL who was kind enough to take my requirements into account and add the filecontentsdefmacro environment. Also a special thanks to Phelype Oleinik who has collaborated and adapted a large part of the code and all LaTeX team for their great work and to the different members of the TeX-SX community who have provided great answers and ideas. Here a note of the main ones:

1. Stack datastructure using LaTeX
2. LaTeX equivalent of ConTeXt buffers
3. Storing an array of strings in a command
4. Collecting contents of environment and store them for later retrieval
5. Collect contents of an environment (that contains verbatim content)

---

[*]This file describes a documentation for v2.2, last revised 2025-03-26.
[†]E-mail: «pablgonz@educarchile.cl».

3 License and Requirements

Permission is granted to copy, distribute and/or modify this software under the terms of the LaTeX Project Public License (lppl), version 1.3 or later (`https://www.latex-project.org/lppl.txt`). The software has the status "maintained".

The The SCONTENTS package is written (mostly) using expl3, it requires an updated version of LaTeX to work (minimum version 2022-06-01). This package can be used with `plain`, `context`, `xelatex`, `lualatex`, `pdflatex` and the classical workflow `latex»dvips»ps2pdf`.

4 The scontents package

4.1 Installation

The package SCONTENTS is present in TeX Live and MiKTeX, use the package manager to install. For manual installation, download scontents.zip and unzip it, run `luatex scontents.ins` and move all files to appropriate locations, then run `mktexlsr`. To produce the documentation with source code run `luatex scontents.ins` and `lualatex scontents.dtx` three times.

```
scontents.tex        »  TDS:tex/generic/scontents/
scontents-code.tex   »  TDS:tex/generic/scontents/
scontents.sty        »  TDS:tex/latex/scontents/
t-scontents.mkiv     »  TDS:tex/context/third/scontents/
scontents.pdf        »  TDS:doc/latex/scontents/
README.md            »  TDS:doc/latex/scontents/
scontents.dtx        »  TDS:source/latex/scontents/
scontents.ins        »  TDS:source/latex/scontents/
```

4.2 Loading and options

The package is loaded in the usual way:

For LaTeX users

```
\usepackage{scontents}
```

or

```
\usepackage[⟨key = val⟩]{scontents}
```

The package options are not available for plain TeX and ConTeXt, see 4.4.

For plain TeX users

```
\input scontents.tex
```

For ConTeXt users

```
\usemodule{scontents}
```

💣 ConTeXt users should use `-luatex`, the implementation does not support LuaMetaTeX.

4.3 The TAB character

Some users use horizontal TABs "⭾" from keyboard to indented the source code of the document and depending on the text editor used, some will use real TABs ("*hard tabs*"), others with "*soft tabs*"(␣␣ or ␣␣␣␣) or both.

At first glance it may seem the same, but the way in which TABs ("*hard tabs*") are processed according to the context in which they are found within a file, both in ⟨*reading*⟩[1] and ⟨*writing*⟩[2] are different and may have adverse consequences.

In a standard LaTeX document, the character TAB "⭾" are treated as explicit spaces (in most contexts) and is the behavior when ⟨*stored contents*⟩, but when ⟨*writing files*⟩ these are preserved.

With a TeX Live distribution, the TAB character is "*printable*" for `latex`, `pdflatex` and `lualatex`, but if you use `xelatex` you must add the `-8bit` option on the command line, otherwise you will get TeX-TAB (`^^I`) in the ⟨*output file*⟩.

As a general recommendation "Do not use TAB character unless strictly necessary", for example within

---

[1]Check the answer given by Ulrich Diez in Keyboard TAB character in argument v (xparse).
[2]Check the answer given by Enrico Gregorio in How to output a tabulation into a file.

a *verbatim* environment that supports this character such as `Verbatim` of the package `fancyvrb` or `lstlisting` of the package listings or when you want to generate a `MakeFile` file.

4.4 Configuration of the options

Most of the options can be passed directly to the package or using the command `\setupsc`. All boolean keys can be passed using the equal sign "=" or just the name of the key, all unknown keys will return an error. In this section are described some of the options, a summary of all options is shown in section 4.5.

`\setupsc`   `\setupsc{⟨keyval list⟩}`

The command `\setupsc` sets the ⟨*keys*⟩ in a global way, it can be used both in the preamble and in the body of the document as many times as desired. However, options set in the declaration of an environment (with `\newenvsc`) have precedence over options set with `\setupsc`.

Options in the optional arguments of environments and commands have the highest precedence, overriding both options in `\newenvsc`, and in `\setupsc`.

`verb-font = {⟨font family⟩}`   default: `\ttfamily`

Sets the ⟨*font family*⟩ used to display the ⟨*stored content*⟩ for the `\typestored` and `\meaningsc` commands. This key is only available as a package option or using `\setupsc`.

`store-all = {⟨seq name⟩}`   default: *not used*

It is a ⟨*meta-key*⟩ that sets the `store-env` key of the `scontents` environment and the `store-cmd` key of the `\Scontents` command. This key is only available as a package option or using `\setupsc`.

`overwrite = {⟨true | false⟩}`   default: *false*

Sets whether the ⟨*files*⟩ generated by `write-out` and `write-env` from the `scontents` environment will be rewritten. This key is available as a package option, for `\setupsc`, for `\Scontents*` and for the environment `scontents`.

`print-all = {⟨true | false⟩}`   default: *false*

It is a ⟨*meta-key*⟩ that sets the `print-env` key of the `scontents` environment and the `print-cmd` key of the `\Scontents` command. This key is only available as a package option or using `\setupsc`.

`force-eol = {⟨true | false⟩}`   default: *false*

Sets if the end of line for the ⟨*stored content*⟩ is hidden or not. This key is necessary only if the last line is the closing of some environment defined by the `fancyvrb` package as `\end{Verbatim}` or another environment that does not support a comments "%" after closing `\end{«env»}%`. This key is available for the `scontents` environment and the `\Scontents` command.

`width-tab = {⟨integer⟩}`   default: *1*

Sets the equivalence in ⟨*spaces*⟩ for the character `TAB` used when displaying stored content in *verbatim style*. The value must be a ⟨*positive integer*⟩. This key is available for the `\typestored` and the `\meaningsc` commands.

4.5 Options Overview

Summary of available options:

key	package	\setupsc	scontents	\Scontents	\Scontents*	\typestored	\meaningsc
store-env	✓	✓	✓	✗	✗	✗	✗
store-cmd	✓	✓	✗	✓	✓	✗	✗
print-env	✓	✓	✓	✗	✗	✗	✗
print-cmd	✓	✓	✗	✓	✓	✗	✗
print-all	✓	✓	✗	✗	✗	✗	✗
store-all	✓	✓	✗	✗	✗	✗	✗
write-env	✗	✗	✓	✗	✗	✗	✗
write-cmd	✗	✗	✗	✗	✓	✗	✗
write-out	✗	✗	✓	✗	✓	✗	✗
overwrite	✓	✓	✓	✗	✓	✗	✗
width-tab	✓	✓	✗	✗	✗	✓	✓
force-eol	✓	✓	✓	✓	✓	✗	✗
verb-font	✓	✓	✗	✗	✗	✗	✗

5 User interface

The user interface consists in `scontents` environment, `\Scontents` and `\Scontents*` commands to ⟨*stored content*⟩ and `\getstored` command to get the ⟨*stored content*⟩ along with other utilities described in this documentation.

5.1 The environment scontents

scontents

```
\begin{scontents}[⟨keyval list⟩]
        ⟨env contents⟩
\end{scontents}
```

The scontents environment allows you to ⟨store⟩ and ⟨write⟩ content, including *verbatim* material. After the package has been loaded, the environment can be used both in the preamble and in the body of the document.

For the correct operation \begin{scontents} and \end{scontents} must be in different lines, all ⟨keys⟩ must be passed separated by commas and "without separation" of the start of the environment.

Comments "%" or "any character" after \begin{scontents} or [⟨keyval list⟩] on the same line are not supported, the package will return an "error" message if this happens. In a similar way comments "%" or "any character" after \end{scontents} on the same line the package will return a "warning" message.

The environment can be ⟨nested⟩ if it is properly balanced and does not appear "literally" in commented lines or in some *verbatim* environment or command. As an example:

```
\begin{scontents}[store-env=outer]
This text is in the outer environment (before nested).
\begin{scontents}[store-env=inner]
This text is found in the inner environment (inside of nested).
\end{scontents}
This text is in the outer environment (after nested).
\end{scontents}
```

Of course, content stored in the ⟨inner⟩ sequence is only available after content stored in the ⟨outer⟩ sequence one has been retrieved, either by using the key print-env or \getstored command.

It is advisable to store content within sequences with different names, so as not to get lost in the order in which content is stored.

Notes for plain TeX and ConTeXt users

In plain TeX there is not environments as in LaTeX. Instead of using the environment scontents, one should use a *pseudo environment* delimited by \scontents and \endscontents.

\scontents
\endscontents

```
\scontents[⟨keyval list⟩]
        ⟨env contents⟩
\endscontents
```

ConTeXt users should use \startscontents and \stopscontents.

\startscontents
\stopscontents

```
\startscontents[⟨keyval list⟩]
        ⟨env contents⟩
\stopscontents
```

Options for environment

The environment options can be configured globally using option in package or the \setupsc command and locally using [⟨key = val⟩] in the environment. The key force-eol is available for this environment.

store-env = {⟨seq name⟩}   default: *contents*

Sets the name of the ⟨sequence⟩ in which the contents will be stored. If the sequence does not exist, it will be created globally.

print-env = {⟨true | false⟩}   default: *false*

Sets if the ⟨stored content⟩ is displayed or not at the time of running the environment. The content is extracted from the ⟨sequence⟩ in which it is stored.

write-env = {⟨file.ext⟩}   default: *not used*

Sets the name of the ⟨external file⟩ in which the ⟨contents⟩ of the environment will be written. The ⟨file.ext⟩ will be created in the working directory, relative or absolute paths are not supported. If ⟨file.ext⟩ does not exist, it will be created or overwritten if the overwrite key is used.

The characters TABs will be written in ⟨file.ext⟩ and the ⟨contents⟩ will be stored in the ⟨sequence⟩ established at that time. XeLaTeX users using the TAB character must add -8bit at the command line, otherwise you will get TeX-TAB (^^I) in ⟨file.ext⟩.

write-out = {⟨file.ext⟩}   default: *not used*

Sets the name of the ⟨external file⟩ in which the ⟨contents⟩ of the environment will be written. The ⟨file.ext⟩ will be created in the working directory, relative or absolute paths are not supported. If ⟨file.ext⟩ does not exist, it will be created or overwritten if the overwrite key is used.

The characters TABs will be written in ⟨*file.ext*⟩, the rest of the ⟨*keys*⟩ will not be available and the ⟨*contents*⟩ will NOT be stored in any ⟨*sequence*⟩. XLATEX users using the TAB character must add -8bit at the command line, otherwise you will get TEX-TAB (^^I) in ⟨*file.ext*⟩.

5.2 The command \newenvsc

\newenvsc | \newenvsc{⟨*env name*⟩}[⟨*initial keys*⟩]

The command \newenvsc allows you to create ⟨*new environments*⟩ based on the same characteristics of the scontents environment. The values entered in [⟨*initial keys*⟩] will be considered as the default values for this new environment and the valid ⟨*keys*⟩ are store-env and print-env. For example:

```
\newenvsc{myenvstore}[store-env=myseq,print-env=false]
```

created the myenvstore environment that stored the content in the myseq sequence and will not display the content when it is executed.

5.3 The command \Scontents

\Scontents | \Scontents[⟨*key = val*⟩]{⟨*argument*⟩}
\Scontents*[⟨*key = val*⟩]{⟨*argument*⟩}
\Scontents*[⟨*key = val*⟩]⟨*del*⟩⟨*argument*⟩⟨*del*⟩

The \Scontents command reads the {⟨*argument*⟩} in standard mode. It is not possible to pass environments such as *verbatim*, but it is possible to use the implementation of \Verb provided by the fvextra package for contents on one line and \lstinline from listings package, but it is preferable to use the starred (*) version.

The \Scontents* command reads the {⟨*argument*⟩} under *verbatim* category code regimen. If its first delimiter is a brace, it will be assumed that the {⟨*argument*⟩} is nested into braces. Otherwise it will be assumed that the ending of that ⟨*argument*⟩ is delimited by that first delimiter ⟨*del*⟩ like command \verb.

Blank lines are preserved, escaped braces "\{" and "\}" must also be balanced if the argument is used with braces and TABs characters typed from the keyboard are converted into spaces. The starred argument (*) and [⟨*key = val*⟩] must not be separated by horizontal spaces between them and the command.

Both versions can be used anywhere in the document and cannot be used as an ⟨*argument*⟩ for other command.

Options for command

The command options can be configured globally using option in package or the \setupsc command and locally using [⟨*key = val*⟩]. The key force-eol is available for this command.

store-cmd = {⟨*seq name*⟩} | default: *contents*

Sets the name of the ⟨*sequence*⟩ in which the contents will be stored. If the sequence does not exist, it will be created globally.

print-cmd = {⟨*true* | *false*⟩} | default: *false*

Sets if the ⟨*stored content*⟩ is displayed or not at the time of running the command. The content is extracted from the ⟨*sequence*⟩ in which it is stored.

Options only for the starred version

write-cmd = {⟨*file.ext*⟩} | default: *not used*

Sets the name of the ⟨*external file*⟩ in which the ⟨*contents*⟩ of the {⟨*argument*⟩} will be written. The ⟨*file.ext*⟩ will be created in the working directory, relative or absolute paths are not supported. If ⟨*file.ext*⟩ does not exist, it will be created or overwritten if the overwrite key is used.

The characters TABs will be written in ⟨*file.ext*⟩ and the ⟨*contents*⟩ will be stored in the ⟨*sequence*⟩ established at that time. XLATEX users using the TAB character must add -8bit at the command line, otherwise you will get TEX-TAB (^^I) in ⟨*file.ext*⟩.

write-out = {⟨*file.ext*⟩} | default: *not used*

Sets the name of the ⟨*external file*⟩ in which the ⟨*contents*⟩ of the {⟨*argument*⟩} will be written. The ⟨*file.ext*⟩ will be created in the working directory, relative or absolute paths are not supported. If ⟨*file.ext*⟩ does not exist, it will be created or overwritten if the overwrite key is used.

The characters TABs will be written in ⟨*file.ext*⟩, the rest of the ⟨*keys*⟩ will not be available and the ⟨*contents*⟩ will NOT be stored in any ⟨*sequence*⟩. XLATEX users using the TAB character must add -8bit at the command line, otherwise you will get TEX-TAB (^^I) in ⟨*file.ext*⟩.

The key overwrite is available for this command.

5.4 The command \getstored

\getstored `\getstored[⟨index⟩]{⟨seq name⟩}`

The command `\getstored` gets the content stored in {⟨seq name⟩} according to the ⟨index⟩ in which it was stored. The command is robust and can be used as an ⟨argument⟩ for another command. If the optional argument is not passed, the default value is the "last element" stored in {⟨seq name⟩}.

5.5 The command \foreachsc

\foreachsc `\foreachsc[⟨key = val⟩]{⟨seq name⟩}`

The command `\foreachsc` goes through and executes the command `\getstored` on the contents stored in {⟨seq name⟩}. If you pass without options run `\getstored` on all contents stored in {⟨seq name⟩}.

Options for command

sep = {⟨code⟩}   default: *empty*

Establishes the separation between each content stored in {⟨seq name⟩}. For example, you can use `sep={\\[10pt]}` for vertical separation of stored contents.

step = {⟨integer⟩}   default: *1*

Sets the increment (⟨step⟩) applied to the value set by key `start` for each element stored in the {⟨seq name⟩}. The value must be a ⟨positive integer⟩.

start = {⟨integer⟩}   default: *1*

Sets the ⟨index⟩ number of the {⟨seq name⟩} from which execution will start. The value must be a ⟨positive integer⟩.

stop = {⟨integer⟩}   default: *total*

Sets the ⟨index⟩ number of the {⟨seq name⟩} from which execution it will finish executing. The value must be a ⟨positive integer⟩.

before = {⟨code⟩}   default: *empty*

Sets the {⟨code⟩} that will be executed ⟨before⟩ each content stored in {⟨seq name⟩}. The {⟨code⟩} must be passed between braces.

after = {⟨code⟩}   default: *empty*

Sets the {⟨code⟩} that will be executed ⟨after⟩ each content stored in {⟨seq name⟩}. The {⟨code⟩} must be passed between braces.

wrapper = {⟨code {#1} more code⟩}   default: *empty*

Wraps the content stored in {⟨seq name⟩} referenced by {#1}. The {⟨code⟩} must be passed between braces. For example `\foreachsc[wrapper={\makebox[1em][l]{#1}}]{contents}`.

5.6 The command \typestored

\typestored `\typestored[⟨index, 1-end, width-tab = number⟩]{⟨seq name⟩}`

The command `\typestored` internally places the content stored in the {⟨seq name⟩} into the `verbatimsc` environment. The ⟨index⟩ corresponds to the position in which the content is stored in the {⟨seq name⟩}, if ⟨1-end⟩ is used "all" content stored in {⟨seq name⟩} will be printed.

If the optional argument is not passed it defaults to the first element stored in the {⟨seq name⟩}. The key `width-tab` is available for this command.

5.7 The command \mergesc

\mergesc `\mergesc[⟨typestored | meaningsc, keys⟩]{⟨{⟨seq A⟩}[⟨index⟩], {⟨seq B⟩}[⟨start - stop⟩], {⟨seq C⟩}[⟨1-end⟩]⟩}`

The command `\mergesc` internally assembles the content stored in the {⟨seq A⟩}[1], {⟨seq B⟩}[2-5] and {⟨seq C⟩}[1-end] into a temporary internal ⟨seq temp⟩.

The use of the keys `typestored` or `meaningsc` are "*mandatory*" and disjoint from each other, the rest of the accepted ⟨keys⟩ are `print-cmd`, `write-out`, `width-tab` and `overwrite`.

The use of the `write-out` key with this command follows the same rules already described, the main advantage is that it allows to join stored content *without rewriting* the file over and over again, by design TEX does not have an append mode for writing files, this effectively allows you to write chunks of code and then merge them into a single file.

5.8 The environment `verbatimsc`

`verbatimsc`

Internal environment used by `\typestored` to display ⟨*verbatim style*⟩ contents.

One consideration to keep in mind is that this is a "*representation*" of the ⟨*stored content*⟩ in a "*verbatim*" environment.

The `verbatimsc` environment can be customized in the following ways after loading the SCONTENTS package:

Using the package fancyvrb:

```
\makeatletter
\let\verbatimsc\@undefined
\let\endverbatimsc\@undefined
\makeatother
\usepackage{fancyvrb}
\DefineVerbatimEnvironment{verbatimsc}{Verbatim}{numbers=left}
```

Using the package minted:

```
\makeatletter
\let\verbatimsc\@undefined
\let\endverbatimsc\@undefined
\makeatother
\usepackage{minted}
\newminted{tex}{linenos}
\newenvironment{verbatimsc}{\VerbatimEnvironment\begin{texcode}}{\end{texcode}}
```

Using the package listings:

```
\makeatletter
\let\verbatimsc\@undefined
\let\endverbatimsc\@undefined
\makeatother
\usepackage{listings}
\lstnewenvironment{verbatimsc}
  {
   \lstset{
          basicstyle=\small\ttfamily,
          columns=fullflexible,
          language=[LaTeX]TeX,
          numbers=left,
          numberstyle=\tiny\color{gray},
          keywordstyle=\color{red}
         }
  }{}
```

6 Other commands provided

6.1 The command `\meaningsc`

`\meaningsc`

`\meaningsc[`⟨*index, width-tab = number*⟩`]{`⟨*seq name*⟩`}`

The command `\meaningsc` executes `\meaning` on the content stored in {⟨*seq name*⟩}. The ⟨*index*⟩ corresponds to the position in which the content is stored in the {⟨*seq name*⟩}.

If the optional argument is not passed it defaults to the first element stored in the {⟨*seq name*⟩}. The key `width-tab` is available for this command.

6.2 The command `\countsc`

`\countsc`

`\countsc{`⟨*seq name*⟩`}`

The command `\countsc` count a number of contents stored in {⟨*seq name*⟩}.

6.3 The command `\cleanseqsc`

`\cleanseqsc`

`\cleanseqsc{`⟨*seq name*⟩`}`

The command `\cleanseqsc` remove all contents stored in {⟨*seq name*⟩}.

7 The SCONTENTS package in action

Remember the abstract on the first page?, this is it:

Abstract

This package allows to store LaTeX code, including "*verbatim*", in ⟨*sequences*⟩ using the l3seq module of expl3. The ⟨*stored content*⟩ can be used as many times as desired in the document, additionally you can write to ⟨*external files*⟩ or show it in ⟨*verbatim style*⟩.

And the description of the package?

The SCONTENTS package allows to ⟨*store contents*⟩ in ⟨*sequences*⟩ or ⟨*external files*⟩. In some ways it is similar to the filecontentsdef package, with the difference in which the ⟨*content*⟩ is stored. The idea behind this package is to get an approach to ConTeXt "*buffers*" by making use ⟨*sequences*⟩.

I've only written:

```
\begin{abstract}
This package allows to store \hologo{LaTeX} code, including \enquote{\emph{verbatim}},
in \mymeta{sequences} using the \mypkg{l3seq} module of \mypkg{expl3}. The \mymeta{stored
content} can be used as many times as desired in the document, additionally you can write
to \mymeta{external files} or show it in \mymeta{verbatim style}.
\end{abstract}
```

and

```
The \mypkg*{scontents} package allows to \mymeta{store contents} in \mymeta{sequences}
or \mymeta{external files}. In some ways it is similar to the \mypkg{filecontentsdef}
package, with the difference in which the \mymeta{content} is stored. The idea behind
this package is to get an approach to \hologo{ConTeXt} \enquote{\emph{buffers}} by
making use \mymeta{sequences}.
```

Of course, I didn't copy and paste. The real code they were written with is:

```
1  \begin{scontents}[store-env=abstract,print-env=true]
2  \begin{abstract}
3  This package allows to store \hologo{LaTeX} code, including \enquote{\emph{verbatim}},
4  in \mymeta{sequences} using the \mypkg{l3seq} module of \mypkg{expl3}. The \mymeta{stored
5  content} can be used as many times as desired in the document, additionally you can write
6  to \mymeta{external files} or show it in \mymeta{verbatim style}.
7  \end{abstract}
8  \end{scontents}
```

and

```
1  \begin{scontents}[store-env=description, print-env=true]
2  The \mypkg*{scontents} package allows to \mymeta{store contents} in \mymeta{sequences}
3  or \mymeta{external files}. In some ways it is similar to the \mypkg{filecontentsdef}
4  package, with the difference in which the \mymeta{content} is stored. The idea behind
5  this package is to get an approach to \hologo{ConTeXt} \enquote{\emph{buffers}} by
6  making use \mymeta{sequences}.
7  \end{scontents}
```

I stored the content in memory and then ran `\getstored` and `\typestored`. This is one of the ways you can use SCONTENTS.

8 Examples

These are some adapted examples that have served as inspiration for the creation of this package. The examples are attached to this documentation and can be extracted from your PDF viewer or from the command line by running:

```
$ pdfdetach -saveall scontents.pdf
```

and then you can use the excellent arara[3] tool to compile them.

8.1 From answers package

Example 1

Adaptation of example 1 of the package answers 📄.

---

[3]The cool TeX automation tool: https://www.ctan.org/pkg/arara

```
1  % arara: pdflatex
2  % arara: clean: { extensions: [ aux, log] }
3  \documentclass{article}
4  \usepackage[store-cmd=solutions]{scontents}
5  \newtheorem{ex}{Exercise}
6  \setlength{\parindent}{0pt}
7  \pagestyle{empty}
8  \begin{document}
9  \section{Problems}
10 \begin{ex}
11 First exercise
12 \Scontents{First solution.}
13 \end{ex}
14
15 \begin{ex}
16 Second exercise
17 \Scontents{Second solution.}
18 \end{ex}
19
20 \section{Solutions}
21 \foreachsc[sep={\\[10pt]}]{solutions}
22 \end{document}
```

8.2   From `filecontentsdef` package

Example 2

Adaptation of example from package filecontentsdef 📄.

```
1  % arara: pdflatex
2  % arara: clean: { extensions: [ aux, log] }
3  \documentclass{article}
4  \usepackage[store-env=defexercise,store-cmd=defexercise]{scontents}
5  \setlength{\parindent}{0pt}
6  \pagestyle{empty}
7  \begin{document}
8  % not starred
9  \Scontents{
10 Prove that \[x^n+y^n=z^n\] is not solvable in positive integers if $n$ is at
11 most $-3$.\par
12 }
13 % starred
14 \Scontents*|Refute the existence of black holes in less than $140$ characters.|
15 % write environment to \jobname.txt
16 \begin{scontents}[write-env=\jobname.txt]
17 \def\NSA{NSA}%
18 Prove that factorization is easily done via probabilistic algorithms and
19 advance evidence from knowledge of the names of its employees in the
20 seventies that the \NSA\ has known that for 40 years.\par
21 \end{scontents}
22 % see all stored
23 \begin{itemize}
24 \foreachsc[before={\item }]{defexercise}
25 \end{itemize}
26 % \getstored are robust :)
27 \section{\getstored[2]{defexercise}}
28 \end{document}
```

8.3   From TeX-SX

Example 3

Adapted from LaTeX equivalent of ConTeXt buffers 📄.

```
1  % arara: pdflatex
2  % arara: clean: { extensions: [ aux, log] }
3  \documentclass{article}
4  \usepackage[store-cmd=tikz]{scontents}
5  \usepackage{tikz}
6  \setlength{\parindent}{0pt}
7  \pagestyle{empty}
8  \Scontents{\matrix{ \node (a) {$a$} ; & \node (b) {$b$} ; \\ } ;}
```

```
 9  \Scontents{\matrix[ampersand replacement=\&]
10  { \node (a) {$a$} ; \& \node (b) {$b$} ; \\ } ;}
11  \Scontents{\matrix{\node (a) {$a$} ; & \node (b) {$b$} ; \\ } ; }
12  \begin{document}
13  \section{tikzpicture}
14  \begin{tikzpicture}
15  \getstored[1]{tikz}
16  \end{tikzpicture}
17
18  \begin{tikzpicture}
19  \getstored[2]{tikz}
20  \end{tikzpicture}
21
22  \begin{tikzpicture}
23  \getstored{tikz}
24  \end{tikzpicture}
25
26  \begin{scontents}[store-env=buffer]
27  Hello World!
28
29  This is a \verb*|fake poor man's buffer :)|.
30  \end{scontents}
31
32  \section{source tikz}
33  \typestored[1]{tikz}
34  \typestored[2]{tikz}
35  \typestored[3]{tikz}
36
37  \section{fake buffer}
38  \subsection{real content}
39  \getstored[1]{buffer}
40  \subsection{verbatim style}
41  \typestored[1]{buffer}
42  \subsection{meaning}
43  \meaningsc[1]{buffer}
44
45  \section{tikz again}
46  \foreachsc[before={\begin{tikzpicture}},after={\end{tikzpicture}},sep={\\[10pt]}]{tikz}
47  \end{document}
```

Example 4

Adapted from Collecting contents of environment and store them for later retrieval 📄.

```
 1  % arara: pdflatex
 2  % arara: clean: { extensions: [ aux, log] }
 3  \documentclass{article}
 4  \usepackage{scontents}
 5  \setlength{\parindent}{0pt}
 6  \pagestyle{empty}
 7  \begin{document}
 8  \begin{scontents}[store-env=main]
 9  Something for main A.
10  \end{scontents}
11
12  \begin{scontents}[store-env=main]
13  Something for \verb|main B|.
14  \end{scontents}
15
16  \begin{scontents}[store-env=other]
17  Something for \verb|other|.
18  \end{scontents}
19
20  \textbf{Let's print them}
21
22  This is first stored in main: \getstored[1]{main}\par
23  This is second stored in main: \getstored{main}\par
24  This is stored in other: \getstored{other}
25
26  \textbf{Print all of stored in main}\par
27  \foreachsc[sep={\\[10pt]}]{main}
28  \end{document}
```

Example 5

Adapted from Collect contents of an environment (that contains verbatim content) 📄.

```
1  % arara: pdflatex
2  % arara: clean: { extensions: [ aux, log] }
3  \documentclass{article}
4  \usepackage{scontents}
5  \setlength{\parindent}{0pt}
6  \pagestyle{empty}
7  \begin{document}
8  \section{Problem stated the first time}
9  \begin{scontents}[print-env=true,store-env=problem]
10 This is normal text.
11 \verb|This is from the verb command.|
12 \verb*|This is from the verb* command.|
13 This is normal text.
14 \begin{verbatim}
15 This is from the verbatim environment:
16 &%{}~
17 \end{verbatim}
18 \end{scontents}
19 \section{Problem restated}
20 \getstored[1]{problem}
21 \section{Problem restated once more}
22 \getstored[1]{problem}
23 \end{document}
```

Example 6

Adapted from Environment hiding its content 📄.

```
1  % arara: pdflatex
2  % arara: clean: { extensions: [ aux, log] }
3  \documentclass[10pt]{article}
4  \usepackage{scontents}
5  \newenvsc{forshort}[store-env=forshort,print-env=false]
6  \setlength{\parindent}{0pt}
7  \pagestyle{empty}
8  \begin{document}
9
10 Something in the whole course.
11
12 \begin{forshort}
13     Just a summary...
14 \end{forshort}
15
16 \end{document}
```

8.4 Customization of `verbatimsc`

Example 7

Customization of `verbatimsc` using the fancyvrb and tcolorbox package 📄.

```
1  % arara: pdflatex
2  % arara: clean: { extensions: [ aux, log] }
3  \documentclass{article}
4  \usepackage{scontents}
5  \makeatletter
6  \let\verbatimsc\@undefined
7  \let\endverbatimsc\@undefined
8  \makeatother
9  \usepackage{fvextra}
10 \usepackage{xcolor}
11 \definecolor{mygray}{gray}{0.9}
12 \usepackage{tcolorbox}
13 \newenvironment{verbatimsc}%
14 {\VerbatimEnvironment
15 \begin{tcolorbox}[colback=mygray, boxsep=0pt, arc=0pt, boxrule=0pt]
16 \begin{Verbatim}[fontsize=\scriptsize, breaklines, breakafter=*, breaksymbolsep=0.5em,
17 breakaftersymbolpre={\,\tiny\ensuremath{\rfloor}}]%
```

```latex
18  {\end{Verbatim}%
19  \end{tcolorbox}}
20  \setlength{\parindent}{0pt}
21  \pagestyle{empty}
22  \begin{document}
23  \section{Test \texttt{\textbackslash begin\{scontents\}} with \texttt{fancyvrb}}
24  Test \verb+{scontents}+ \par
25
26  \begin{scontents}
27  Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+
28  with index 1.
29
30  Prove new \Verb*{ fancyvrb with braces } and environment \verb+Verbatim*+
31  \begin{verbatim}
32   verbatim  environment
33  \end{verbatim}
34  \end{scontents}
35
36  \section{Test \texttt{\textbackslash Scontents} with \texttt{fancyvrb}}
37  \Scontents{ We have coded this in \LaTeX: $E=mc^2$.}
38
39  \section{Test \texttt{\textbackslash getstored}}
40  \getstored[1]{contents}\par
41  \getstored{contents}
42
43  \section{Test \texttt{\textbackslash meaningsc}}
44  \meaningsc[1]{contents}\par
45  \meaningsc[2]{contents}
46
47  \section{Test \texttt{\textbackslash typestored}}
48  \typestored[1]{contents}
49  \typestored[2]{contents}
50  \end{document}
```

Example 8

Customization of `verbatimsc` using the listings package 📄.

```latex
1   % arara: pdflatex
2   % arara: clean: { extensions: [ aux, log] }
3   \documentclass{article}
4   \usepackage{scontents}
5   \makeatletter
6   \let\verbatimsc\@undefined
7   \let\endverbatimsc\@undefined
8   \makeatother
9   \usepackage{xcolor}
10  \usepackage{listings}
11  \lstnewenvironment{verbatimsc}
12    {
13    \lstset{
14          basicstyle=\small\ttfamily,
15          breaklines=true,
16          columns=fullflexible,
17          language=[LaTeX]TeX,
18          numbers=left,
19          numbersep=1em,
20          numberstyle=\tiny\color{gray},
21          keywordstyle=\color{red}
22      }
23    }{}
24  \setlength{\parindent}{0pt}
25  \pagestyle{empty}
26  \begin{document}
27  \section{Test \texttt{\textbackslash begin\{scontents\}} with \texttt{listings}}
28  Test \verb+{scontents}+ \par
29
30  \begin{scontents}
31  Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+ with index 1.\par
32
33  Prove \lstinline[basicstyle=\ttfamily]| lstinline | and environment \verb+Verbatim*+
34  \begin{verbatim}
```

```
35    verbatim  environment
36  \end{verbatim}
37  \end{scontents}
38
39  \section{Test \texttt{\textbackslash Scontents*} with \texttt{listings}}
40
41  \Scontents*+ We have coded this in \lstinline[basicstyle=\ttfamily]|\LaTeX: $E=mc^2$|
42  and more.+
43
44  \section{Test \texttt{\textbackslash getstored}}
45  \getstored{contents}\par
46  \getstored[1]{contents}
47
48  \section{Test \texttt{\textbackslash typestored}}
49  \typestored[1]{contents}
50  \typestored[2]{contents}
51  \end{document}
```

Example 9

Customization of `verbatimsc` using the minted package 📄.

```
1   % arara: xelatex: {shell: true, options: [-8bit]}
2   % arara: clean: { extensions: [ aux, log] }
3   \documentclass{article}
4   \usepackage{scontents}
5   \makeatletter
6   \let\verbatimsc\@undefined
7   \let\endverbatimsc\@undefined
8   \makeatother
9   \usepackage{minted}
10  \newminted{tex}{linenos}
11  \newenvironment{verbatimsc}{\VerbatimEnvironment\begin{texcode}}{\end{texcode}}
12  \pagestyle{empty}
13  \setlength{\parindent}{0pt}
14  \begin{document}
15  \section{Test \texttt{\textbackslash begin\{scontents\}} with \texttt{minted}}
16  Test \verb+{scontents}+ \par
17
18  \begin{scontents}[overwrite,write-env=\jobname.tsc,force-eol=true]
19  Using \verb+scontents+ env no \verb+[key=val]+, save in seq \verb+contents+
20  with index 1.\par
21
22  Prove new \Verb*{ new fvextra with braces } and environment \verb+Verbatim*+
23  \begin{Verbatim}[obeytabs, showtabs, tab=\rightarrowfill, tabcolor=red]
24  No tab
25        One real tab
26              Two real Tab plus        one tab
27  \end{Verbatim}
28  \end{scontents}
29
30  \section{See \Verb{\jobname.tsc}}
31  Read \Verb{\jobname.tsc} (shows TABs as red arrows):
32  \VerbatimInput[obeytabs, showtabs, tab=\rightarrowfill, tabcolor=red]{\jobname.tsc}
33
34  \section{Test \texttt{\textbackslash Scontents} with \texttt{minted}}
35
36  \Scontents{ We have coded \par this in \LaTeX: $E=mc^2$.}
37
38  \section{Test \texttt{\textbackslash getstored}}
39  \getstored[1]{contents}\par
40  \getstored{contents}
41
42  \section{Test \texttt{\textbackslash typestored}}
43  \typestored[1]{contents}
44  \typestored[2]{contents}
45  \end{document}
```

8.5 The command \mergesc in action

The command \mergesc in action, adapted from Denis Bitouzé request at https://github.com/pablgonz/scontents/issues/2 📄.

```
1  % arara: pdflatex
2  % arara: clean: { extensions: [ aux, log ] }
3  \documentclass{article}
4  \usepackage{scontents}
5  % Fix part of a MCE that should go before babel's loading
6  \begin{scontents}[store-env=mce]
7  \documentclass[french]{article}
8  \usepackage[T1]{fontenc}
9  \usepackage[utf8]{inputenc}
10 \usepackage{lmodern}
11 \usepackage[a4paper]{geometry}
12 \end{scontents}
13 % Fix part of a MCE that should go after (>=) babel's loading
14 \begin{scontents}[store-env=mce]
15 \usepackage{babel}
16 \begin{document}
17 \end{scontents}
18 % Fix part of a MCE that should go after its body
19 \begin{scontents}[store-env=mce]
20 \end{document}
21 \end{scontents}
22 \begin{document}
23 \section{First annswer}
24 % Variable part of a MCE that should added to the fixed preamble, before babel's loading
25 \begin{scontents}[store-env=mce-1]
26 \usepackage{amsmath}
27 \end{scontents}
28 % Variable part of a MCE being the code snippet
29 \begin{scontents}[store-env=mce-1]
30 \begin{align}
31   0 & \neq 1 \\
32   1 & \neq 0
33 \end{align}
34 \end{scontents}
35 \begin{description}
36 \item[Preamble's addition]\leavevmode
37   \typestored[1]{mce-1}
38 \item[Code snippet]\leavevmode
39   \typestored[2]{mce-1}
40 \item[MCE]\leavevmode
41   \mergesc[typestored, print-cmd=true]
42     {
43       {mce}[1], {mce-1}[1], {mce}[2], {mce-1}[2], {mce}[3]
44     }
45 \end{description}
46 \section{Second annswer}
47 % Variable part of a MCE that should added to the fixed preamble, before babel's loading
48 \begin{scontents}[store-env=mce-2]
49 \usepackage{amsmath}
50 \end{scontents}
51 % Variable part of a MCE being the code snippet
52 \begin{scontents}[store-env=mce-2]
53 \begin{flalign}
54   0 & \neq 1 \\
55   1 & \neq 0
56 \end{flalign}
57 \end{scontents}
58
59 \begin{description}
60 \item[Preamble's addition]\leavevmode
61   \typestored[1]{mce-2}
62 \item[Code snippet]\leavevmode
63   \typestored[2]{mce-2}
64 \item[MCE]\leavevmode
65   \mergesc[typestored, print-cmd=true, write-out=mce.txt, overwrite=true]
66     {
67       {mce}[1], {mce-2}[1], {mce}[2], {mce-2}[2], {mce}[3]
68     }
69 \end{description}
70 \end{document}
```

9   Change history

In this section you will find some (not all) of the changes in ꜱᴄᴏɴᴛᴇɴᴛꜱ development, from the first public implementation using the filecontentsdef package to the current version with only expl3.

v2.2 (ctan), 2025-03-26	– Fix internal definition for some functions.
	– Replace `\peek_charcode_ignore_spaces:NTF` by `\peek_charcode:NTF`.
	– Set correct code for `\obeyedline` implement in LATEX release 2024-06-01.
v2.1 (ctan), 2024-06-14	– Fix `\cleansc` command.
	– Add `\mergesc` command.
	– Fix internal definition for seq var.
	– Fix internal code for `\typestored`.
	– Replace `\cs_argument_spec:N` by `\cs_parameter_spec:N`.
	– Detect l3keys2e package (obsolete in june 2022 LATEX release).
	– Minor adjustments in the documentation.
v2.0 (ctan), 2022-04-04	– Adapting the verbatimsc environment (compatibility verbatim package).
	– Removed compatibility layer for older LATEX releases.
	– Fix loader in plain TEX and ConTEXt.
	– Minor adjustments in the documentation.
v1.9 (ctan), 2020-01-21	– Update and improvements in the internal code.
	– Updating the generic code for I/O verification.
	– Add `write-cmd` and `write-out` keys for `\Scontents*`.
	– Fix sep key in `\foreachsc`.
v1.8 (ctan), 2019-11-18	– Add `\newenvsc` command.
	– Fix nested environment in plain TEX and ConTEXt.
	– Modified default value in `\getstored`.
	– Add `overwrite` key to reduce I/O operations.
	– Deleted an unnecessary group in the code.
v1.7 (ctan), 2019-10-29	– The verbatimsc environment was rewritten.
	– Minor adjustments in documentation.
v1.6 (ctan), 2019-10-26	– The internal behavior of `\getstored` has been modified.
	– The internal behavior of `\foreachsc` has been modified.
	– Corrected file extension for ConTEXt.
	– Remove spurious warning.
v1.5 (ctan), 2019-10-24	– Add support for plain TEX and ConTEXt.
	– Split internal code for optimization.
	– Add support for vertical spaces in `key=val`.
	– Add `\foreachsc` command.
	– Check if verbatim package is loaded.
v1.4 (ctan), 2019-10-03	– Add `store-all` key.
	– Messages and keys were separated.
	– Restructuring of documentation.
	– Now the version of expl3 is checked instead of xparse.
	– The internal behavior of `force-eol` has been modified.
v1.3 (ctan), 2019-09-24	– The environment can now nest.
	– Added `force-eol`, `verb-font` and `width-tab` keys.
	– The extra space has been removed when you run `\getstored`.
	– Internal code has been rewritten more efficiently.
	– Remove starred argument for `\typestored`.
	– Remove filecontentsdef dependency.
	– Changing `\regex_replace_all:` for `\tl_replace_all:`.
v1.2 (ctan), 2019-08-28	– Restructuring of documentation.
	– Added copy of `\tex_scantokens:`.
v1.1 (ctan), 2019-08-12	– Extension of documentation.
	– Replace `\tex_endinput:D` by `\file_input_stop:`.
v1.0 (ctan), 2019-07-30	– First public release.

10 Index of Documentation

The italic numbers denote the pages where the corresponding entry is described.

11 References

[1] The LATEX Project. "The expl3 package". Available from CTAN, https://www.ctan.org/pkg/expl3, 2023.

[2] The LATEX Project. "The xparse package". Available from CTAN, https://www.ctan.org/pkg/xparse, 2023.

[3] The LATEX Project. "The l3keys2e package". Available from CTAN, https://www.ctan.org/pkg/l3keys2e, 2022.

[4] WRIGHT, JOSEPH. "Programming key–value in expl3". Available from TUGBOAT, https://www.tug.org/TUGboat/tb31-1/tb97wright-l3keys.pdf, 2010.

12 Implementation

The most recent publicly released version of SCONTENTS is available at CTAN: https://www.ctan.org/pkg/scontents. Historical and developmental versions are available at ⓞ https://github.com/pablgonz/scontents. While general feedback via email is welcomed, specific bugs or feature requests should be reported through the issue tracker: https://github.com/pablgonz/scontents/issues.

12.1 Declaration of the package

First we set up the module name for l3doc:

```
1  ⟨@@=scontents⟩
```

Now we define some common macros to hold the package date and version:

```
2  ⟨loader⟩\def\ScontentsFileDate{2025-03-26}%
3  ⟨core⟩\def\ScontentsCoreFileDate{2025-03-26}%
4  ⟨*loader⟩
5  \def\ScontentsFileVersion{2.2}%
6  \def\ScontentsFileDescription{Stores LaTeX contents in memory or files}%
```

The LaTeX loader is fairly simple: just load the dependencies, load the core code, and then set interfaces up.

```
7  ⟨*latex⟩
8  \IfFormatAtLeastTF { 2022-06-01 }
9   { }
10  {
11    \RequirePackage{l3keys2e}[2020/02/08]
12    \PackageWarning { scontents }
13     {
14        The next update removes compatibility with versions prior to 2024.
15     }
16  }
17  \ProvidesExplPackage
18   {scontents} {\ScontentsFileDate} {\ScontentsFileVersion} {\ScontentsFileDescription}
19  ⟨/latex⟩
```

The plain TeX and ConTeXt loaders are similar (probably because I don't know how to make a proper ConTeXt module :-). We define a LaTeX-style \ver@scontents.sty macro with version info (just in case) and add \ExplSyntaxOn to be able to load xparse later.

```
20  ⟨*!latex⟩
21  ⟨context⟩\writestatus{loading}{User Module scontents v\ScontentsFileVersion}
22  ⟨context⟩\unprotect
23  \input expl3-generic.tex
24  \ExplSyntaxOn
25  \tl_gset:cx { ver @ scontents . sty } { \ScontentsFileDate\space
26    v\ScontentsFileVersion\space \ScontentsFileDescription }
27  \iow_log:x { Package: ~ scontents ~ \use:c { ver @ scontents . sty } } }
28  ⟨/!latex⟩
```

In plain TeX, check that the package isn't being loaded twice (LaTeX and ConTeXt already defend against that):

```
29  ⟨*plain⟩
30  \msg_set:nnn { scontents } { already-loaded }
31   { The~'scontents'~package~is~already~loaded.~Aborting~input~\msg_line_context:. }
32  \cs_if_exist:NT \__scontents_rescan_tokens:n
33   {
34    \msg_warning:nn { scontents } { already-loaded }
35    \ExplSyntaxOff
36    \file_input_stop:
37   }
38  ⟨/plain⟩
```

12.2 Definition of variables by format

We define and set variables that must be handled separately in order to work properly with plain TeX, ConTeXt and LaTeX.

\g__scontents_end_verbatimsc_tl   A global token list \g__scontents_end_verbatimsc_tl match when ending verbatimsc environment.

```
39  \tl_new:N \g__scontents_end_verbatimsc_tl
40  \tl_gset_rescan:Nnn \g__scontents_end_verbatimsc_tl
```

```
41      {
42        \char_set_catcode_other:N \\
43  ⟨∗latex⟩
44        \char_set_catcode_other:N \{
45        \char_set_catcode_other:N \}
46  ⟨/latex⟩
47      }
48  ⟨latex⟩   { \end{verbatimsc} }
49  ⟨plain⟩   { \endverbatimsc }
50  ⟨context⟩  { \stopverbatimsc }
```

(*End of definition for* `\g__scontents_end_verbatimsc_tl`.)

`\c__scontents_end_env_tl`
`\l__scontents_env_name_tl`

A token list `\c__scontents_end_env_tl` match when ending environments defined by `\newenvsc`, `\l__scontents_env_name_tl` storing the name of environments defined by `\newenvsc`.

```
51  \tl_new:N \l__scontents_env_name_tl
52  \tl_const:Nx \c__scontents_end_env_tl
53      {
54        \c_backslash_str
55  ⟨latex|plain⟩    end
56  ⟨context⟩      stop
57  ⟨latex⟩    \c_left_brace_str
58          \exp_not:N \l__scontents_env_name_tl
59  ⟨latex⟩    \c_right_brace_str
60      }
```

(*End of definition for* `\c__scontents_end_env_tl` *and* `\l__scontents_env_name_tl`.)

Now we load the core SCONTENTS code:

```
61  \file_input:n { scontents-code.tex }
```

`\__scontents_format_case:nnn`

Sometimes we need to detect the format from within a macro:

```
62  \cs_new:Npn \__scontents_format_case:nnn #1 #2 #3
63  ⟨latex⟩   {#1} % LaTeX
64  ⟨plain⟩   {#2} % Plain/Generic
65  ⟨context⟩  {#3} % ConTeXt
```

(*End of definition for* `\__scontents_format_case:nnn`.)

Checking that the package was loaded with the proper loader code. This code was copied from `expl3-code.tex`.

```
66  ⟨/loader⟩
67  ⟨∗core⟩
68  \begingroup
69    \catcode32=10
70    \endlinechar=32
71    \def\next{\endgroup}%
72    \expandafter\ifx\csname PackageError\endcsname\relax
73      \begingroup
74        \def\next{\endgroup\endgroup}%
75        \def\PackageError#1#2#3%
76          {%
77             \endgroup
78             \errhelp{#3}%
79             \errmessage{#1 Error: #2!}%
80          }%
81    \fi
82    \expandafter\ifx\csname ScontentsFileDate\endcsname\relax
83      \def\next
84        {%
85          \PackageError{scontents}{No scontents loader detected}
86            {%
87               You have attempted to use the scontents code directly rather than using
88               the correct loader. Loading of scontents will abort.
89            }%
90          \endgroup
91          \endinput
92        }%
93    \else
```

```
94    \ifx\ScontentsFileDate\ScontentsCoreFileDate
95    \else
96      \def\next
97        {%
98          \PackageError{scontents}{Mismatched scontents files detected}
99            {%
100              You have attempted to load scontents with mismatched files:
101              probably you have one or more files 'locally installed' which
102              are in conflict. Loading of scontents will abort.
103            }%
104          \endgroup
105          \endinput
106        }%
107    \fi
108  \fi
109 \next
```

12.3   Definition of temporary variables

\l__scontents_macro_tmp_tl

\l__scontents_temp_tl

\g__scontents_temp_tl

\l__scontents_tmpa_int

\l__scontents_temp_bool

The token list \l__scontents_macro_tmp_tl is a temporary token list to hold the contents of the macro/environment. \l__scontents_temp_tl, \g__scontents_temp_tl, \l__scontents_tmpa_int and \l__scontents_temp_bool are generic temporary vars.

```
110 \tl_new:N \l__scontents_macro_tmp_tl
111 \tl_new:N \l__scontents_temp_tl
112 \tl_new:N \g__scontents_temp_tl
113 \int_new:N \l__scontents_tmpa_int
114 \bool_new:N \l__scontents_temp_bool
```

(*End of definition for* \l__scontents_macro_tmp_tl *and others.*)

\l__scontents_keys_tl

Stores unused ⟨*keys*⟩ to be forwarded to other commands.

```
115 \tl_new:N \l__scontents_keys_tl
```

(*End of definition for* \l__scontents_keys_tl.)

12.4   Compatibility layer with plain TEX and ConTEXt

When loading the package outside of LATEX we can't usually use xparse. However since xparse now ltcmd is part of the LATEX kernel is loadable in any format.

```
116 ⟨/core⟩
117 ⟨*loader&!latex⟩
118 \int_set:Nn \l__scontents_tmpa_int { \char_value_catcode:n { `\@ } }
119 \char_set_catcode_letter:N \@
120 \file_input:n { xparse-generic.tex }
121 \char_set_catcode:nn { `\@ } { \l__scontents_tmpa_int }
122 ⟨/loader&!latex⟩
123 ⟨*core⟩
```

12.5   Definition of keys for the package

We create some common ⟨*keys*⟩ that will be used by the options passed to the package as well as by the environments and commands defined.

```
124 \keys_define:nn { scontents }
125   {
126     store-env .tl_set:N         = \l__scontents_name_seq_env_tl,
127     store-env .initial:n        = contents,
128     store-env .value_required:n = true,
129     store-cmd .tl_set:N         = \l__scontents_name_seq_cmd_tl,
130     store-cmd .initial:n        = contents,
131     store-cmd .value_required:n = true,
132     verb-font .tl_set:N         = \l__scontents_verb_font_tl,
133     verb-font .value_required:n = true,
134     print-env .bool_set:N       = \l__scontents_print_env_bool,
135     print-env .initial:n        = false,
136     print-env .default:n        = true,
137     print-cmd .bool_set:N       = \l__scontents_print_cmd_bool,
138     print-cmd .initial:n        = false,
139     print-cmd .default:n        = true,
```

```
140      force-eol .bool_set:N        = \l__scontents_forced_eol_bool,
141      force-eol .initial:n         = false,
142      force-eol .default:n         = true,
143      overwrite .bool_set:N        = \l__scontents_overwrite_bool,
144      overwrite .initial:n         = false,
145      overwrite .default:n         = true,
146      width-tab .int_set:N         = \l__scontents_tab_width_int,
147      width-tab .initial:n         = 1,
148      width-tab .value_required:n  = true,
149      print-all .meta:n            = { print-env = #1 , print-cmd = #1 },
150      print-all .default:n         = true,
151      store-all .meta:n            = { store-env = #1 , store-cmd = #1 },
152      store-all .value_required:n  = true
153    }
154 ⟨/core⟩
155 ⟨loader⟩\keys_define:nn { scontents }
156 ⟨latex⟩   { verb-font .initial:n = \ttfamily }
157 ⟨plain|context⟩  { verb-font .initial:n = \tt }
```

In LaTeX mode we load l3keys2e process the ⟨*keys*⟩ as options passed on to the package, the package l3keys2e will verify the ⟨*keys*⟩ and will return an error when they are

```
158 ⟨*latex⟩
159 \IfFormatAtLeastTF { 2022-06-01 }
160   {
161     \ProcessKeyOptions [ scontents ]
162   }
163   { \ProcessKeysOptions { scontents } }
164 ⟨/latex⟩
165 ⟨*core⟩
```

12.6   Internal variables and utility functions

\l__scontents_fname_out_tl
\l__scontents_every_line_env_tl
\l__scontents_file_iow

The token list \l__scontents_fname_out_tl is used for store the name of the ⟨*output file*⟩, when there's one. Its value is set by the keys write-env, write-out and write-cmd.

The token list \l__scontents_every_line_env_tl holds the contents of an environment, scontents by default, as it's being read. \l__scontents_file_iow is an output stream for saving the contents of an environment (or command) to a file.

This variables is used by the function __scontents_file_tl_write_start:n (see 12.10.5).

```
166 \tl_new:N \l__scontents_fname_out_tl
167 \tl_new:N \l__scontents_every_line_env_tl
168 \iow_new:N \l__scontents_file_iow
```

(*End of definition for* \l__scontents_fname_out_tl, \l__scontents_every_line_env_tl, *and* \l__scontents_file_iow.)

\l__scontents_foreach_name_seq_tl
\l__scontents_foreach_before_tl
\l__scontents_foreach_after_tl

\l__scontents_foreach_name_seq_tl is the name assigned to the sequence on which the loop will be made, \l__scontents_foreach_before_tl and \l__scontents_foreach_after_tl are token lists in which the assigned material will be placed before and after the execution of the \foreachsc loop.

```
169 \tl_new:N \l__scontents_foreach_name_seq_tl
170 \tl_new:N \l__scontents_foreach_before_tl
171 \tl_new:N \l__scontents_foreach_after_tl
```

(*End of definition for* \l__scontents_foreach_name_seq_tl, \l__scontents_foreach_before_tl, *and* \l__scontents_foreach_after_tl.)

\l__scontents_env_nesting_int
\l__scontents_foreach_stop_int

\l__scontents_seq_item_seq stores the indexes in the sequence of the items requested to \typestored or \meaningsc. \l__scontents_env_nesting_int stores the current nesting level of the scontents environment. \l__scontents_foreach_stop_int will save the value at which the \foreachsc loop will stop.

```
172 \int_new:N \l__scontents_foreach_stop_int
173 \seq_new:N \l__scontents_seq_item_seq
174 \int_new:N \l__scontents_env_nesting_int
```

(*End of definition for* \l__scontents_env_nesting_int *and* \l__scontents_foreach_stop_int.)

\l__scontents_writing_bool
\l__scontents_storing_bool
\l__scontents_writable_bool

The boolean \l__scontents_writing_bool keeps track of whether we should write to a file, and \l__scontents_storing_bool determines whether it is in write-only mode when the key write-out is used.

```
175 \bool_new:N \l__scontents_writing_bool
176 \bool_set_false:N \l__scontents_writing_bool
177 \bool_new:N \l__scontents_storing_bool
178 \bool_set_true:N  \l__scontents_storing_bool
179 \bool_new:N \l__scontents_writable_bool
```

(*End of definition for* \l__scontents_writing_bool, \l__scontents_storing_bool, *and* \l__scontents_writable_bool.)

\l__scontents_foreach_before_bool
\l__scontents_foreach_after_bool
\l__scontents_foreach_stop_bool
\l__scontents_foreach_wrapper_bool

Boolean variables used by the \foreachsc loop.

```
180 \bool_new:N \l__scontents_foreach_before_bool
181 \bool_set_false:N \l__scontents_foreach_before_bool
182 \bool_new:N \l__scontents_foreach_after_bool
183 \bool_set_false:N \l__scontents_foreach_after_bool
184 \bool_new:N \l__scontents_foreach_stop_bool
185 \bool_set_false:N \l__scontents_foreach_stop_bool
186 \bool_new:N \l__scontents_foreach_wrapper_bool
187 \bool_set_false:N \l__scontents_foreach_wrapper_bool
```

(*End of definition for* \l__scontents_foreach_before_bool *and others.*)

\l__scontents_foreach_print_seq

The \l__scontents_foreach_print_seq is the sequence used by \foreachsc.

```
188 \seq_new:N \l__scontents_foreach_print_seq
189 \seq_new:c { g__scontents_name_sc!internal_seq }
```

(*End of definition for* \l__scontents_foreach_print_seq.)

\c__scontents_hidden_space_str

\c__scontents_hidden_space_str is a constant *string* to used to hide the ⟨*forced space*⟩ added by TeX when recording content in a macro. This *string* contains the *reserved phrase* "%^^Ascheol%" which is added to the end of the argument stored in seq when the key force-eol is false.

```
190 \str_const:Nx \c__scontents_hidden_space_str
191   { \c_percent_str \c_circumflex_str \c_circumflex_str A scheol \c_percent_str }
```

(*End of definition for* \c__scontents_hidden_space_str.)

\q__scontents_stop
\q__scontents_mark

Some quarks used along the code as macro delimiters.

```
192 \quark_new:N \q__scontents_stop
193 \quark_new:N \q__scontents_mark
```

(*End of definition for* \q__scontents_stop *and* \q__scontents_mark.)

\s__scontents_stop
\s__scontents_mark

```
194 \scan_new:N \s__scontents_stop
195 \scan_new:N \s__scontents_mark
```

(*End of definition for* \s__scontents_stop *and* \s__scontents_mark.)

\l__scontents_cur_seq_name_str

```
196 \str_new:N \l__scontents_cur_seq_name_str
```

(*End of definition for* \l__scontents_cur_seq_name_str.)

__scontents_use_i_delimit_by_s_stop:nw
__scontents_use_none_delimit_by_s_stop:w

```
197 \cs_new:Npn \__scontents_use_delimit_by_s_stop:nw #1 \s__scontents_stop {#1}
198 \cs_new:Npn \__scontents_use_i_delimit_by_s_stop:nw #1 #2 \s__scontents_stop {#1}
199 \cs_new:Npn \__scontents_use_none_delimit_by_s_stop:w #1 \s__scontents_stop { }
```

(*End of definition for* __scontents_use_i_delimit_by_s_stop:nw *and* __scontents_use_none_delimit_by_s_stop:w.)

\l__scontents_save_sf_int
\l__scontents_save_skip

Internal variables used by functions __scontents_bsphack: and __scontents_esphack:.

```
200 \int_new:N \l__scontents_save_sf_int
201 \skip_new:N \l__scontents_save_skip
```

(End of definition for `\l__scontents_save_sf_int` *and* `\l__scontents_save_skip`*.)*

`\__scontents_rescan_tokens:n`
`\__scontents_rescan_tokens:x`
`\__scontents_rescan_tokens:V`

The function `\tl_rescan:nn` provided by expl3 doesn't fit the needs of this package because it does not allow catcode changes inside the argument, so verbatim commands used inside one of SCONTENTS's commands/environments will not work. Here we create a private copy of `\tex_scantokens:D` which will serve our purposes. See the answer by Ulrich Diez in How do use {<setup>} in \tl_set_rescan:Nnn to replace \scantokens?

```
202 \cs_new_protected:Npn \__scontents_rescan_tokens:n #1 { \tex_scantokens:D {#1} }
203 \cs_generate_variant:Nn \__scontents_rescan_tokens:n { V, x }
```

(End of definition for `\__scontents_rescan_tokens:n`*.)*

`\__scontents_tab:`
`\__scontents_par:`

Control sequences to replace tab (`^^I`) and form feed (`^^L`) characters.

```
204 \cs_new:Npx \__scontents_tab: { \c_space_tl }
205 \cs_new:Npn \__scontents_par: { ^^J ^^J }
```

(End of definition for `\__scontents_tab:` *and* `\__scontents_par:`*.)*

`\tl_remove_once:NV`
`\tl_replace_all:Nxx`
`\tl_replace_all:Nxn`
`\tl_replace_all:Nnx`
`\tl_if_empty:fTF`

Some nonstandard kernel variants.

```
206 \cs_generate_variant:Nn \tl_remove_once:Nn { NV }
207 \cs_generate_variant:Nn \tl_replace_all:Nnn { Nx, Nxx, Nnx }
208 \cs_generate_variant:Nn \msg_error:nnnn { nnx }
209 \prg_generate_conditional_variant:Nnn \tl_if_empty:n { f } { p, TF }
```

(End of definition for `\tl_remove_once:NV`*,* `\tl_replace_all:Nxx`*, and* `\tl_if_empty:fTF`*.)*

12.7   Defining keys for the environment and commands

We add the ⟨*keys*⟩ divided into subgroups to handle errors and *unknown* ⟨*keys*⟩ separately.

12.7.1   Keys for environment scontents

We define a set of ⟨*keys*⟩ for environment scontents.

```
210 \keys_define:nn { scontents / scontents }
211   {
212     write-env .code:n         = {
213                                   \bool_set_true:N \l__scontents_writing_bool
214                                   \tl_set:Nn \l__scontents_fname_out_tl {#1}
215                                 },
216     write-out .code:n         = {
217                                   \bool_set_false:N \l__scontents_storing_bool
218                                   \bool_set_true:N  \l__scontents_writing_bool
219                                   \tl_set:Nn \l__scontents_fname_out_tl {#1}
220                                 },
221     write-env .value_required:n = true,
222     write-out .value_required:n = true,
223     print-env .meta:nn        = { scontents } { print-env = #1 },
224     print-env .default:n      = true,
225     store-env .meta:nn        = { scontents } { store-env = #1 },
226     force-eol .meta:nn        = { scontents } { force-eol = #1 },
227     force-eol .default:n      = true,
228     overwrite .meta:nn        = { scontents } { overwrite = #1 },
229     overwrite .default:n      = true,
230     unknown   .code:n         = { \__scontents_parse_environment_keys:n {#1} }
231   }
```

12.7.2   Keys for command \Scontents

We define a set of ⟨*keys*⟩ for commands \Scontents and \Scontents*.

```
232 \keys_define:nn { scontents / Scontents }
233   {
234     write-cmd .code:n         = {
235                                   \bool_set_true:N \l__scontents_writing_bool
236                                   \tl_set:Nn \l__scontents_fname_out_tl {#1}
237                                 },
238     write-out .code:n         = {
239                                   \bool_set_false:N \l__scontents_storing_bool
```

```
240                                 \bool_set_true:N  \l__scontents_writing_bool
241                                 \tl_set:Nn \l__scontents_fname_out_tl {#1}
242                             },
243     write-cmd .value_required:n = true,
244     write-out .value_required:n = true,
245     print-cmd .meta:nn          = { scontents } { print-cmd = #1 },
246     print-cmd .default:n        = true,
247     store-cmd .meta:nn          = { scontents } { store-cmd = #1 },
248     force-eol .meta:nn          = { scontents } { force-eol = #1 },
249     force-eol .default:n        = true,
250     overwrite .meta:nn          = { scontents } { overwrite = #1 },
251     overwrite .default:n        = true,
252     unknown   .code:n           = { \__scontents_parse_command_keys:n {#1} }
253   }
```

12.7.3   Keys for command `\foreachsc`

We define a set of ⟨keys⟩ for command `\foreachsc`.

```
254 \keys_define:nn { scontents / foreachsc }
255   {
256     before  .code:n         = {
257                                 \bool_set_true:N \l__scontents_foreach_before_bool
258                                 \tl_set:Nn \l__scontents_foreach_before_tl {#1}
259                             },
260     before  .value_required:n = true,
261     after   .code:n         = {
262                                 \bool_set_true:N \l__scontents_foreach_after_bool
263                                 \tl_set:Nn \l__scontents_foreach_after_tl {#1}
264                             },
265     after   .value_required:n = true,
266     start   .int_set:N        = \l__scontents_foreach_start_int,
267     start   .value_required:n = true,
268     start   .initial:n        = 1,
269     stop    .code:n         = {
270                                 \bool_set_true:N \l__scontents_foreach_stop_bool
271                                 \int_set:Nn \l__scontents_foreach_stop_int {#1}
272                             },
273     stop    .value_required:n = true,
274     step    .int_set:N        = \l__scontents_foreach_step_int,
275     step    .value_required:n = true,
276     step    .initial:n        = 1,
277     wrapper .code:n         = {
278                                 \bool_set_true:N \l__scontents_foreach_wrapper_bool
279                                 \cs_set_protected:Npn
280                                 \__scontents_foreach_wrapper:n ##1 {#1}
281                             },
282     wrapper .value_required:n = true,
283     sep     .tl_set:N         = \l__scontents_foreach_sep_tl,
284     sep     .initial:n        = {},
285     sep     .value_required:n = true,
286     unknown .code:n           = { \__scontents_parse_foreach_keys:n {#1} }
287   }
```

12.7.4   Key for commands `\typestored` and `\meaningsc`

We define a ⟨key⟩ for command `\typestored` and `\meaningsc`. Both commands accept the same type of optional arguments, just define a common ⟨key⟩.

```
288 \bool_new:N \l__scontents_print_aux_bool
289 \bool_set_true:N  \l__scontents_print_aux_bool
290 \keys_define:nn { scontents / typemeaning }
291   {
292     width-tab .meta:nn   = { scontents } { width-tab = #1 },
293     write-out .code:n    = {
294                             \bool_set_false:N \l__scontents_storing_bool
295                             \bool_set_true:N  \l__scontents_writing_bool
296                             \tl_set:Nn \l__scontents_fname_out_tl {#1}
297                         },
298     overwrite .meta:nn   = { scontents } { overwrite = #1 },
299     overwrite .default:n = true,
300     unknown   .code:n    = { \__scontents_parse_type_meaning_key:n {#1} }
```

```
301    }
```

12.8   Handling undefined keys

The ⟨keys⟩ are stored in the string variable \l_keys_key_str, and the value (if any) is passed as an argument to each ⟨function⟩.

12.8.1   Undefined keys for environment scontents

__scontents_parse_environment_keys:n
__scontents_parse_environment_keys:nn

We check the ⟨keys⟩ passed to the environment scontents and process it with __scontents_parse_-environment_keys:n if the ⟨key⟩ is *unknown* we return an error message.

```
302  \cs_new_protected:Npn \__scontents_parse_environment_keys:n #1
303    { \exp_args:NV \__scontents_parse_environment_keys:nn \l_keys_key_str {#1} }
304  \cs_new_protected:Npn \__scontents_parse_environment_keys:nn #1#2
305    {
306      \tl_if_blank:nTF {#2}
307        { \msg_error:nnn { scontents } { env-key-unknown } {#1} }
308        { \msg_error:nnnn { scontents } { env-key-value-unknown } {#1} {#2} }
309    }
```

(*End of definition for* __scontents_parse_environment_keys:n *and* __scontents_parse_environment_keys:nn.)

12.8.2   Undefined keys for \Scontents and \Scontents*

__scontents_parse_command_keys:n
__scontents_parse_command_keys:nn

We check the ⟨keys⟩ passed to commands \Scontents or \Scontents* and process it with __scontents_-parse_command_keys:n if the ⟨key⟩ is *unknown* we return an error message.

```
310  \cs_new_protected:Npn \__scontents_parse_command_keys:n #1
311    { \exp_args:NV \__scontents_parse_command_keys:nn \l_keys_key_str {#1} }
312  \cs_new_protected:Npn \__scontents_parse_command_keys:nn #1#2
313    {
314      \tl_if_blank:nTF {#2}
315        { \msg_error:nnn { scontents } { cmd-key-unknown } {#1} }
316        { \msg_error:nnnn { scontents } { cmd-key-value-unknown } {#1} {#2} }
317    }
```

(*End of definition for* __scontents_parse_command_keys:n *and* __scontents_parse_command_keys:nn.)

12.8.3   Undefined keys for \foreachsc

__scontents_parse_foreach_keys:n
__scontents_parse_foreach_keys:nn

We check the ⟨keys⟩ passed to command \foreachsc and process it with __scontents_parse_-foreach_keys:n, if the ⟨key⟩ is *unknown* we return an error message.

```
318  \cs_new_protected:Npn \__scontents_parse_foreach_keys:nn #1#2
319    {
320      \tl_if_blank:nTF {#2}
321        { \msg_error:nnn { scontents } { for-key-unknown } {#1} }
322        { \msg_error:nnnn { scontents } { for-key-value-unknown } {#1} {#2} }
323    }
324  \cs_new_protected:Npn \__scontents_parse_foreach_keys:n #1
325    { \exp_args:NV \__scontents_parse_foreach_keys:nn \l_keys_key_str {#1} }
```

(*End of definition for* __scontents_parse_foreach_keys:n *and* __scontents_parse_foreach_keys:nn.)

12.8.4   Undefined keys for \typestored and \meaningsc

__scontents_parse_type_meaning_key:n
__scontents_parse_type_meaning_key:nn

The commands \typestored and \meaningsc accept an optional argument for setting the width-tab to print the stored contents. However their optional argument also contains the number of the item to retrieve from the stored sequence. To avoid the awkward \typestored[][⟨options⟩]{...} syntax, we'll make the commands have a single optional argument which is processed by l3keys, and the unknown keys are brought here to __scontents_parse_typemeaning_key:n to process.

First we check if the ⟨key⟩ is an integer using \int_to_roman:n. If it is, we check that the value passed to the key is blank (otherwise something odd as 1=1 might have been used). If everything is correct, then set the value of the integer which holds the ⟨index⟩. Otherwise raise an error about an *unknown* option.

```
326  \cs_new_protected:Npn \__scontents_parse_type_meaning_key:n #1
327    { \exp_args:NV \__scontents_parse_type_meaning_key:nn \l_keys_key_str {#1} }
328  \cs_new_protected:Npn \__scontents_parse_type_meaning_key:nn #1#2
329    {
330      \tl_if_blank:nTF {#2}
331        { \__scontents_parse_type_meaning_range:w #1 - \q__scontents_mark - \s__scontents_mark }
```

```
332          { \msg_error:nnnn { scontents } { type-key-value-unknown } {#1} {#2} }
333      }
334  \cs_new_protected:Npn \__scontents_parse_type_meaning_range:w #1 - #2 - #3 \s__scontents_mark
335      {
336          \__scontents_range_parser:nnxn {#1} {#2}
337              { \seq_count:c { g__scontents_name_\l__scontents_cur_seq_name_str _seq } }
338              { \msg_error:nnn { scontents } { type-key-unknown } }
339      }
340  \cs_new_protected:Npn \__scontents_range_parser:nnnn #1 #2 #3 #4
341      {
342          \exp_args:Nxx \__scontents_range_parser_aux:nnn
343              { \str_if_eq:nnTF {#1} { end } {#3} { \exp_not:n {#1} } }
344              { \str_if_eq:nnTF {#2} { end } {#3} { \exp_not:n {#2} } }
345              { #4 }
346      }
347  \cs_generate_variant:Nn \__scontents_range_parser:nnnn { nnxn }
348  \cs_new_protected:Npn \__scontents_range_parser_aux:nnn #1 #2 #3
349      {
350          \__scontents_tl_if_head_is_q_mark:nTF {#2}
351              {
352                  \tl_if_empty:fTF { \int_to_roman:n { -0 #1 } }
353                      { \seq_put_right:Nx \l__scontents_seq_item_seq { \int_eval:n {#1} } }
354                      { #3 {#1} }
355              }
356              {
357                  \bool_lazy_and:nnTF
358                      { \tl_if_empty_p:f { \int_to_roman:n { -0 #1 } } }
359                      { \tl_if_empty_p:f { \int_to_roman:n { -0 #2 } } }
360                      {
361                      \int_compare:nNnTF {#2} > {#1}
362                          { \int_step_inline:nnnn {#1} { 1 } {#2} }
363                          { \int_step_inline:nnnn {#1} { -1 } {#2} }
364                              { \seq_put_right:Nn \l__scontents_seq_item_seq {##1} }
365                      }
366                      { #3 { #1-#2 } }
367              }
368      }
```

(*End of definition for __scontents_parse_type_meaning_key:n and __scontents_parse_type_meaning_key:nn.*)

12.9 Programming of the sequences

The storage of the package is done using seq variables. Here we set up the macros that will manage the variables.

__scontents_append_contents:nn
__scontents_append_contents:Vx

The function __scontents_append_contents:nn creates a seq variable if one didn't exist and appends the contents in the argument to the right of the sequence.

```
369  \cs_new_protected:Npn \__scontents_append_contents:nn #1#2
370      {
371          \tl_if_blank:nT {#1}
372              { \msg_error:nn { scontents } { empty-store-cmd } }
373          \seq_if_exist:cF { g__scontents_name_#1_seq }
374              { \seq_new:c { g__scontents_name_#1_seq } }
375          \seq_gput_right:cn { g__scontents_name_#1_seq } {#2}
376      }
377  \cs_generate_variant:Nn \__scontents_append_contents:nn { Vx }
```

(*End of definition for __scontents_append_contents:nn.*)

__scontents_getfrom_seq:nn
__scontents_getfrom_seq:Nn
__scontents_getfrom_seq:nnn

The function __scontents_getfrom_seq:nn retrieves the saved item from the sequence.

```
378  \cs_new:Npn \__scontents_getfrom_seq:Nn #1#2
379      {
380          \seq_if_exist:cTF { g__scontents_name_#2_seq }
381              {
382                  \exp_args:Nf \__scontents_getfrom_seq:nNn
383                      { \seq_count:c { g__scontents_name_#2_seq } } #1 {#2}
384              }
385              { \msg_expandable_error:nnn { scontents } { undefined-storage } {#2} }
386      }
```

```
387  \cs_new:Npn \__scontents_getfrom_seq:nNn #1 #2 #3
388    { \seq_map_tokens:Nn #2 { \__scontents_getfrom_seq_aux:nnn {#1} {#3} } }
389  \cs_new:Npn \__scontents_getfrom_seq_aux:nnn #1 #2 #3
390    { \exp_args:Nnf \use:n { \__scontents_getfrom_seq:nnn {#1} } { \int_eval:n {#3} } {#2} }
391  \cs_new:Npn \__scontents_getfrom_seq:nn #1#2
392    {
393      \seq_if_exist:cTF { g__scontents_name_#2_seq }
394        {
395          \exp_args:Nf \__scontents_getfrom_seq:nnn
396            { \seq_count:c { g__scontents_name_#2_seq } }
397            {#1} {#2}
398        }
399        { \msg_expandable_error:nnn { scontents } { undefined-storage } {#2} }
400    }
401  \cs_new:Npn \__scontents_getfrom_seq:nnn #1#2#3
402    {
403      \bool_lazy_or:nnTF
404        { \int_compare_p:nNn {#2} = { 0 } }
405        { \int_compare_p:nNn { \int_abs:n {#2} } > {#1} }
406        { \msg_expandable_error:nnnnn { scontents } { index-out-of-range } {#2} {#3} {#1} }
407        { \seq_item:cn { g__scontents_name_#3_seq } {#2} }
408    }
```

(*End of definition for* __scontents_getfrom_seq:nn, __scontents_getfrom_seq:Nn, *and* __scontents_getfrom_seq:nnn.)

__scontents_lastfrom_seq:n
__scontents_lastfrom_seq:V

The function __scontents_lastfrom_seq:n retrieves the last saved item from the sequence when \l__scontents_print_env_bool or \l__scontents_print_cmd_bool is true.

```
409  \cs_new_protected:Npn \__scontents_lastfrom_seq:n #1
410    {
411      \tl_gset:Nx \g__scontents_temp_tl { \seq_item:cn { g__scontents_name_#1_seq } {-1} }
412      \group_insert_after:N \__scontents_rescan_tokens:V
413      \group_insert_after:N \g__scontents_temp_tl
414      \group_insert_after:N \tl_gclear:N
415      \group_insert_after:N \g__scontents_temp_tl
416    }
417  \cs_generate_variant:Nn \__scontents_lastfrom_seq:n { V }
```

(*End of definition for* __scontents_lastfrom_seq:n.)

__scontents_store_to_seq:NN

The function __scontents_store_to_seq:NN writes the recorded contents in #1 to the log and stores it in #2.

```
418  \cs_new_protected:Npn \__scontents_store_to_seq:NN #1#2
419    {
420      \tl_log:N #1
421      \__scontents_append_contents:Vx #2 { \exp_not:V #1 }
422    }
```

(*End of definition for* __scontents_store_to_seq:NN.)

12.10 The command \newenvsc and environment scontents

In order to be able to define environments that behave similarly to scontents, we define a generic environment and make all other environment as wrappers around that one.

12.10.1 The command \newenvsc

\newenvsc
__scontents_env_setting:nn
__scontents_env_define:nnn

The internal function __scontents_env_setting:nn defines two functions __scontents_#1_-env_begin: and __scontents_#1_env_end:, which set the current environment's name in #1 and \l__scontents_env_name_tl and default properties in #2 then call __scontents_setup_verb_-processor:, the generic __scontents_env_generic_begin: and __scontents_env_generic_end:.

Finally the function __scontents_env_define:nnn will create the environments.

```
423  \cs_new_protected:Npn \__scontents_env_setting:nn #1 #2
424    {
425      \cs_new_protected:cpn { __scontents_#1_env_begin: }
426        {
427          \tl_set:Nn \l__scontents_env_name_tl {#1}
428          \keys_set:nn { scontents } {#2}
429          \__scontents_setup_verb_processor:
```

```
430              \__scontents_env_generic_begin:
431            }
432        \cs_new_protected:cpn { __scontents_#1_env_end: }
433          { \__scontents_env_generic_end: }
434        \exp_args:Nooo % http://noooooooooooooooo.com :) jeje
435        \__scontents_env_define:nnn { \tl_to_str:n {#1} }
436          { \cs:w __scontents_#1_env_begin: \cs_end: }
437          { \cs:w __scontents_#1_env_end: \cs_end: }
438      }
```
439 ⟨/core⟩
440 ⟨∗loader⟩
```
441 \NewDocumentCommand \newenvsc { m O{} }
442    {
```
443 ⟨latex|plain⟩   \cs_if_exist:cTF { #1 }
444 ⟨context⟩   \cs_if_exist:cTF { start #1 }
```
445        { \msg_error:nnn { scontents } { env-already-defined } {#1} }
446        { \__scontents_env_setting:nn {#1} {#2} }
447    }
448 \cs_new_protected:Npn \__scontents_env_define:nnn #1 #2 #3
449    {
```
450 ⟨latex|plain⟩   \NewDocumentEnvironment {#1} { }
451 ⟨context⟩   \cs_new_protected:cpn { start #1 }
```
452        {
```
453 ⟨!latex⟩   \group_begin:
```
454          #2
455        }
```
456 ⟨context⟩   \cs_new_protected:cpn { stop #1 }
```
457        {
458          #3
```
459 ⟨!latex⟩   \group_end:
```
460        }
461    }
```
462 ⟨/loader⟩
463 ⟨∗core⟩

(*End of definition for* \newenvsc *,* __scontents_env_setting:nn *, and* __scontents_env_define:nnn*. This function is documented on page 5.*)

12.10.2 Generic definition of the environment

__scontents_env_generic_begin:
__scontents_env_generic_end:

Now we define the generic environment functions __scontents_env_generic_begin: and __scontents_-env_generic_end:.

```
464 \cs_new_protected:Npn \__scontents_env_generic_begin:
465    {
466        \char_set_catcode_active:N \^^M
467        \__scontents_start_environment:w
468    }
469 \cs_new_protected:Npn \__scontents_env_generic_end:
470    {
471        \__scontents_stop_environment:
472        \__scontents_finish_storing:NNN \l__scontents_macro_tmp_tl
473        \l__scontents_name_seq_env_tl \l__scontents_print_env_bool
474    }
```

(*End of definition for* __scontents_env_generic_begin: *and* __scontents_env_generic_end:*.*)

12.10.3 Definition of the environment scontents

scontents
\scontents
\endscontents
\startscontents
\stopscontents

Finally defining the scontents environment should be easy :)

```
475 ⟨/core⟩
476 ⟨loader⟩\newenvsc{scontents}
477 ⟨∗core⟩
```

(*End of definition for* scontents *and others. These functions are documented on page 4.*)

12.10.4 key val for environment

__scontents_grab_optional:n
__scontents_grab_optional:w

The macro __scontents_grab_optional:w is called from the scontents environment with the tokens following the \begin{scontents} when the next character is a [. This function is defined using xparse to exploit its delimited argument processor.

The function is called from a context where `^^M` is active, so `\__scontents_normalise_line_ends:N` is used to replace active `^^M` characters by spaces.

```
478  ⟨/core⟩
479  ⟨∗loader⟩
480  \NewDocumentCommand \__scontents_grab_optional:w { r[] }
481    { \__scontents_grab_optional:n {#1} }
482  ⟨/loader⟩
483  ⟨∗core⟩
484  \cs_new_protected:Npn \__scontents_grab_optional:n #1
485    {
486      \tl_if_novalue:nF {#1}
487        {
488          \tl_set:Nn \l__scontents_temp_tl {#1}
489          \__scontents_normalise_line_ends:N \l__scontents_temp_tl
490          \keys_set:nV { scontents / scontents } \l__scontents_temp_tl
491        }
492      \__scontents_start_after_option:w
493    }
```

(*End of definition for* `\__scontents_grab_optional:n` *and* `\__scontents_grab_optional:w`.)

12.10.5 The environment itself

`\__scontents_start_environment:w`
`\__scontents_start_after_option:w`
`\__scontents_check_line_process:xn`
`\__scontents_stop_environment:`

Here we make `^^I`, `^^L` and `^^M` active characters so that the end of line can be "seen" to be used as a delimiter, and TeX doesn't try to eliminate space-like characters.

First we check if the immediate next token after `\begin{scontents}` is a `[`. If it is, then `\__scontents_-grab_optional:w` is called to do the heavy lifting. `\__scontents_grab_optional:w` processes the optional argument and calls `\__scontents_start_after_option:w`.

The function `\__scontents_start_after_option:w` also checks for trailing tokens after the optional argument and issues an error if any.

In all cases, the function `\__scontents_check_line_process:xn` checks that everything past `\begin{scontents}` is empty and then process the environment.

The function `\__scontents_check_line_process:xn` calls the function `\__scontents_file_tl_-write_start:V` which will then read the contents of the environment and optionally store them in a token list or write to an external file.

When that's done, the function `\__scontents_file_write_stop:N` does the cleanup. This part of the code is inspired and adapted from the code of the package xsimverb by Clemens Niederberger.

```
494  \group_begin:
495    \char_set_catcode_active:N \^^I
496    \char_set_catcode_active:N \^^L
497    \char_set_catcode_active:N \^^M
498  \cs_new_protected:Npn \__scontents_normalise_line_ends:N #1
499    { \tl_replace_all:Nnn #1 { ^^M } { ~ } }
500  \cs_new_protected:Npn \__scontents_start_environment:w #1 ^^M
501    {
502      \tl_if_head_is_N_type:nTF {#1}
503        {
504          \str_if_eq:eeTF { \tl_head:n {#1} } { [ }
505            { \__scontents_grab_optional:w #1 ^^M }
506            { \__scontents_check_line_process:xn { } {#1} }
507        }
508        { \__scontents_check_line_process:xn { } {#1} }
509    }
510  \cs_new_protected:Npn \__scontents_start_after_option:w #1 ^^M
511    { \__scontents_check_line_process:xn { [...] } {#1} }
512  \cs_new_protected:Npn \__scontents_check_line_process:xn #1 #2
513    {
514      \tl_if_blank:nF {#2}
515        {
516          \msg_error:nnxn { scontents } { junk-after-begin }
517            { after~\c_backslash_str begin { \l__scontents_env_name_tl } #1 } {#2}
518        }
519      \__scontents_make_control_chars_active:
520      \__scontents_file_tl_write_start:V \l__scontents_fname_out_tl
521    }
522  \cs_new_protected:Npn \__scontents_stop_environment:
523    {
524      \__scontents_file_write_stop:N \l__scontents_macro_tmp_tl
```

```
525    \bool_lazy_and:nnT
526      { \l__scontents_storing_bool }
527      { \tl_if_empty_p:N \l__scontents_macro_tmp_tl }
528      {
529        \msg_warning:nnx { scontents } { empty-environment } { \l__scontents_env_name_tl }
530      }
531    }
```

(*End of definition for* __scontents_start_environment:w *and others.*)

__scontents_file_tl_write_start:n
__scontents_file_tl_write_start:V
__scontents_verb_processor_iterate:w
__scontents_verb_processor_iterate:nnn
__scontents_setup_verb_processor:
__scontents_file_write_stop:N
__scontents_remove_leading_nl:n
__scontents_remove_leading_nl:w

This is the main macro to collect the contents of a verbatim environment. The macro starts a group, opens the ⟨*output file*⟩, if necessary, sets verbatim catcodes, and then issues ^^M (set equal to __scontents_-ret:w) to read the environment line by line until reaching its end. The output token list will be appended with an active ^^J character and the line just read, and this line is written to the output file, if any. At the end of the environment the ⟨*output file*⟩ is closed (if it was open), and the output token list is smuggled out of the verbatim group. A leading ^^J is removed from the token list using __scontents_remove_-leading_nl:n (which expects an active ^^J token at the head of the token list; a low level TEX error is raised otherwise).

```
532  \cs_new_protected:Npn \__scontents_file_tl_write_start:n #1
533    {
534      \group_begin:
535        \__scontents_file_if_writable:nTF {#1}
536          {
537            \bool_set_true:N \l__scontents_writable_bool
538            \iow_open:Nn \l__scontents_file_iow {#1}
539          }
540          { \bool_set_false:N \l__scontents_writable_bool }
541        \tl_clear:N \l__scontents_every_line_env_tl
542        \seq_map_function:NN \l_char_special_seq \char_set_catcode_other:N
543        \int_step_function:nnnN { 128 } { 1 } { 255 } \char_set_catcode_letter:n
544        \cs_set_protected:Npx \__scontents_ret:w ##1 ^^M
545          {
546            \exp_not:N \__scontents_verb_processor_iterate:w
547            ##1 \c__scontents_end_env_tl
548              \c__scontents_end_env_tl
549              \exp_not:N \q__scontents_stop
550          }
551        \__scontents_make_control_chars_active:
552        \__scontents_ret:w
553    }
554  \cs_new:Npn \__scontents_setup_verb_processor:
555    {
556      \use:x
557        {
558          \cs_set:Npn \exp_not:N \__scontents_verb_processor_iterate:w
559            ####1 \c__scontents_end_env_tl
560            ####2 \c__scontents_end_env_tl
561            ####3 \exp_not:N \q__scontents_stop
562        } { \__scontents_verb_processor_iterate:nnn {##1} {##2} {##3} }
563    }
564  \cs_new:Npn \__scontents_verb_processor_iterate:nnn #1 #2 #3
565    {
566      \tl_if_blank:nTF {#3}
567        {
568          \__scontents_analyse_nesting:n {#1}
569          \__scontents_verb_processor_output:n {#1}
570        }
571        {
572          \__scontents_if_nested:TF
573            {
574              \__scontents_nesting_decr:
575              \__scontents_verb_processor_output:x
576                { \exp_not:n {#1} \c__scontents_end_env_tl \exp_not:n {#2} }
577            }
578            {
579              \tl_if_blank:nF {#1}
580                { \__scontents_verb_processor_output:n {#1} }
581              \cs_set_protected:Npx \__scontents_ret:w
582                {
583                  \__scontents_env_end_function:
```

```
584          \bool_lazy_or:nnF
585            { \tl_if_blank_p:n {#2} }
586            { \str_if_eq_p:ee {#2} { \c_percent_str } }
587            {
588              \str_if_eq:VnF \c__scontents_hidden_space_str {#2}
589                {
590                  \msg_warning:nnnn { scontents } { rescanning-text }
591                    {#2} { \tl_use:N \l__scontents_env_name_tl }
592                }
593              \__scontents_rescan_tokens:n {#2}
594            }
595          }
596        \char_set_active_eq:NN ^^M \__scontents_ret:w
597      }
598    }
599  ^^M
600  }
601 \cs_new:Npn \__scontents_env_end_function:
602  {
603    \__scontents_format_case:nnn
604      { \exp_not:N \end { \if_false: } \fi: }
605      { \exp_after:wN \exp_not:N \cs:w end }
606      { \exp_after:wN \exp_not:N \cs:w stop }
607    \tl_use:N \l__scontents_env_name_tl
608    \__scontents_format_case:nnn
609      { \if_false: { \fi: } }
610      { \cs_end: }
611      { \cs_end: }
612  }
613 \cs_new_protected:Npn \__scontents_file_write_stop:N #1
614  {
615    \bool_if:NT \l__scontents_writable_bool
616      { \iow_close:N \l__scontents_file_iow }
617    \use:x
618      {
619        \group_end:
620        \bool_if:NT \l__scontents_storing_bool
621          {
622            \tl_set:Nn \exp_not:N #1
623              { \exp_args:NV \__scontents_remove_leading_nl:n \l__scontents_every_line_env_tl }
624          }
625      }
626  }
627 \cs_new:Npn \__scontents_remove_leading_nl:n #1
628  {
629    \tl_if_head_is_N_type:nTF {#1}
630      {
631        \exp_args:Nf
632          \__scontents_remove_leading_nl:nn
633            { \tl_head:n {#1} } {#1}
634      }
635      { \exp_not:n {#1} }
636  }
637 \cs_new:Npn \__scontents_remove_leading_nl:nn #1 #2
638  {
639    \token_if_eq_meaning:NNTF ^^J #1
640      { \exp_not:o { \__scontents_remove_leading_nl:w #2 } }
641      { \exp_not:n {#2} }
642  }
643 \cs_new:Npn \__scontents_remove_leading_nl:w ^^J { }
```

(*End of definition for* `\__scontents_file_tl_write_start:n` *and others.*)

`\__scontents_verb_processor_output:n`
`\__scontents_verb_processor_output:x`

The function `\__scontents_verb_processor_output:n` does the output of each line read, to a token list and to a file, depending on the booleans `\l__scontents_writing_bool` and `\l__scontents_storing_-bool`.

```
644 \cs_new_protected:Npn \__scontents_verb_processor_output:n #1
645  {
646    \bool_if:NT \l__scontents_writable_bool
647      { \iow_now:Nn \l__scontents_file_iow {#1} }
```

```
648    \bool_if:NT \l__scontents_storing_bool
649        { \tl_put_right:Nn \l__scontents_every_line_env_tl { ^^J #1 } }
650    }
651  \group_end:
652  \cs_generate_variant:Nn \__scontents_verb_processor_output:n { x }
653  \cs_generate_variant:Nn \__scontents_file_tl_write_start:n { V }
```

(*End of definition for* `\__scontents_verb_processor_output:n`.)

`\__scontents_analyse_nesting:n`
`\__scontents_analyse_nesting:w`
`\__scontents_nesting_decr:`
`\__scontents_use_none_delimit_by_q_stop:w`
`\__scontents_if_nested:`*TF*

`\__scontents_analyse_nesting:n` scans nested `\begin{scontents}` and steps a `\l__scontents_-` `env_nesting_int` counter. The `\__scontents_if_nested:` conditional tests if we're in a nested environment, and `\__scontents_nesting_decr:` reduces the nesting level, if an `\end{scontents}` is found.

Multiple `\end{scontents}` in the same line are not supported...

```
654  \cs_new_protected:Npn \__scontents_analyse_nesting:n #1
655    {
656      \int_zero:N \l__scontents_tmpa_int
657      \__scontents_analyse_nesting_format:n {#1}
658      \int_compare:nNnT { \l__scontents_tmpa_int } > { 1 }
659        { \msg_warning:nn { scontents } { multiple-begin } }
660    }
661  \cs_new_protected:Npn \__scontents_nesting_incr:
662    {
663      \int_incr:N \l__scontents_env_nesting_int
664      \int_incr:N \l__scontents_tmpa_int
665    }
666  \cs_new_protected:Npn \__scontents_nesting_decr:
667    { \int_decr:N \l__scontents_env_nesting_int }
668  \prg_new_protected_conditional:Npnn \__scontents_if_nested: { TF }
669    {
670      \int_compare:nNnTF { \l__scontents_env_nesting_int } > { \c_zero_int }
671        { \prg_return_true: }
672        { \prg_return_false: }
673    }
674  \cs_new:Npn \__scontents_use_none_delimit_by_q_stop:w #1 \q__scontents_stop { }
```

In LaTeX, environments start with `\begin{`«env»`}`, so checking if a string contains `\begin{scontents}` is straightforward. Since no `}` can appear inside «env», then just a macro delimited by `}` is enough.

```
675  \use:x
676    {
677      \cs_new_protected:Npn \exp_not:N \__scontents_analyse_nesting_latex:w ##1
678        \c_backslash_str begin \c_left_brace_str ##2 \c_right_brace_str
679    }    {
680        \__scontents_tl_if_head_is_q_mark:nTF {#2}
681          { \__scontents_use_none_delimit_by_q_stop:w }
682          {
683            \str_if_eq:VnT \l__scontents_env_name_tl {#2}
684              { \__scontents_nesting_incr: }
685            \__scontents_analyse_nesting_latex:w
686          }
687      }
688  \cs_new_protected:Npx \__scontents_analyse_nesting_latex:n #1
689    {
690      \__scontents_analyse_nesting_latex:w #1
691        \c_backslash_str begin
692          \c_left_brace_str \exp_not:N \q__scontents_mark \c_right_brace_str
693      \exp_not:N \q__scontents_stop
694    }
```

In other formats, however, we don't have an "end anchor" to delimit the environment name, so a delimited macro won't help. We have to search for the entire environment command (usually `\scontents` and `\startscontents`).

```
695  \cs_new_protected:Npn \__scontents_analyse_nesting_generic_process:nn #1 #2
696    {
697      \tl_if_head_is_N_type:nTF {#2}
698        {
699          \__scontents_tl_if_head_is_q_mark:nF {#2}
700            {
701              \__scontents_nesting_incr:
```

```
702                       \__scontents_analyse_nesting_generic:w #2 \q__scontents_stop
703                     }
704                 }
705             { \__scontents_analyse_nesting_generic:w #2 \q__scontents_stop }
706     }
707 \cs_new_protected:Npn \__scontents_analyse_nesting_generic:nn #1 #2
708     {
709         \__scontents_define_generic_nesting_function:n {#1}
710         \use:x
711             {
712                 \exp_not:N \__scontents_analyse_nesting_generic:w #2
713                     \c_backslash_str #1 \tl_use:N \l__scontents_env_name_tl
714                         \exp_not:N \q__scontents_mark \exp_not:N \q__scontents_stop
715             }
716     }
717 \cs_new_protected:Npn \__scontents_define_generic_nesting_function:n #1
718     {
719         \use:x
720             {
721                 \cs_set_protected:Npn \exp_not:N \__scontents_analyse_nesting_generic:w ####1
722                     \c_backslash_str #1 \tl_use:N \l__scontents_env_name_tl
723                         ####2 \exp_not:N \q__scontents_stop
724             }   { \__scontents_analyse_nesting_generic_process:nn {##1} {##2} }
725     }
726 ⟨/core⟩
727 ⟨*loader⟩
728 ⟨latex⟩\cs_new_eq:NN \__scontents_analyse_nesting_format:n
729 ⟨latex⟩    \__scontents_analyse_nesting_latex:n
730 ⟨!latex⟩\cs_new_protected:Npn \__scontents_analyse_nesting_format:n
731 ⟨plain⟩    { \__scontents_analyse_nesting_generic:nn { } }
732 ⟨context⟩  { \__scontents_analyse_nesting_generic:nn { start } }
733 ⟨/loader⟩
734 ⟨*core⟩
```

(*End of definition for* __scontents_analyse_nesting:n *and others.*)

12.10.6   Recording of the content in the sequence

__scontents_finish_storing:NNN   Finishes the environment by optionally calling __scontents_store_to_seq: and then clearing the temporary token list.

```
735 \cs_new_protected:Npn \__scontents_finish_storing:NNN #1 #2 #3
736     {
737         \bool_if:NT \l__scontents_storing_bool
738             {
739                 \bool_if:NF \l__scontents_forced_eol_bool
740                     { \tl_put_right:Nx #1 { \c__scontents_hidden_space_str } }
741                 \__scontents_store_to_seq:NN #1 #2
742                 \bool_if:NT #3 { \__scontents_lastfrom_seq:V #2 }
743             }
744     }
745 ⟨/core⟩
```

(*End of definition for* __scontents_finish_storing:NNN.)

12.11   The environment verbatimsc

\verbatimsc   In plain TeX we emulate LaTeX's verbatim environment.
\endverbatimsc
__scontents_verbatimsc_aux:
__scontents_vobeyspaces:
__scontents_xverb:
__scontents_nolig_list:
__scontents_xobeysp:

```
746 ⟨*plain⟩
747 \cs_new_protected:Npn \verbatimsc
748     {
749         \group_begin:
750             \__scontents_verbatimsc_aux: \frenchspacing \__scontents_vobeyspaces:
751             \__scontents_xverb:
752     }
753 \cs_new_protected:Npn \endverbatimsc
754     { \group_end: }
755 \cs_new_protected:Npn \__scontents_verbatimsc_aux:
756     {
757         \skip_vertical:N \parskip
758         \dim_zero:N \parindent
```

```
759    \skip_set:Nn \parfillskip { 0pt plus 1fil }
760    \skip_set:Nn \parskip { 0pt plus0pt minus0pt }
761    \tex_par:D
762    \bool_set_false:N \l__scontents_temp_bool
763    \cs_set:Npn \par
764      {
765        \bool_if:NTF \l__scontents_temp_bool
766          {
767            \mode_leave_vertical:
768            \null
769            \tex_par:D
770            \penalty \interlinepenalty
771          }
772          {
773            \bool_set_true:N \l__scontents_temp_bool
774            \mode_if_horizontal:T
775              { \tex_par:D \penalty \interlinepenalty }
776          }
777      }
778    \cs_set_eq:NN \do \char_set_catcode_other:N
779    \dospecials \obeylines
780    \tl_use:N \l__scontents_verb_font_tl
781    \cs_set_eq:NN \do \__scontents_do_noligs:N
782    \__scontents_nolig_list:
783    \tex_everypar:D \exp_after:wN
784      { \tex_the:D \tex_everypar:D \tex_unpenalty:D }
785    }
786 \cs_new_protected:Npn \__scontents_nolig_list:
787    { \do\`\do\<\do\>\do\,\do\'\do\- }
788 \cs_new_protected:Npn \__scontents_vobeyspaces:
789    { \__scontents_set_active_eq:NN \  \__scontents_xobeysp: }
790 \cs_new_protected:Npn \__scontents_xobeysp:
791    { \mode_leave_vertical: \nobreak \  }
792 ⟨/plain⟩
```

(*End of definition for* \verbatimsc *and others.*)

\dospecials   xparse also requires LATEX's \dospecials. In case it doesn't exist (at the time scontents is loaded) we define \dospecials to use the \l_char_special_seq.

```
793 ⟨*!latex⟩
794 \cs_if_exist:NF \dospecials
795    {
796      \cs_new:Npn \dospecials
797        { \seq_map_function:NN \l_char_special_seq \do }
798    }
799 ⟨/!latex⟩
```

(*End of definition for* \dospecials.)

12.12 The command \Scontents

User command to ⟨*stored content*⟩, adapted from code by Ulrich Diez in Stringify input - \string on token list and code by user siracusa in Convert a macro from Latex2e to expl3

__scontents_bsphack: We emulate \@bsphack and \@esphack for plain TEX. This is necessary to prevent unwanted spaces when
__scontents_esphack: the print-cmd key is false.

```
800 ⟨*core⟩
801 \cs_new_protected:Npn \__scontents_bsphack:
802    {
803      \scan_stop:
804      \mode_if_horizontal:T
805        {
806          \skip_set_eq:NN \l__scontents_save_skip \tex_lastskip:D
807          \int_set_eq:NN \l__scontents_save_sf_int \tex_spacefactor:D
808        }
809    }
810 \cs_new_protected:Npn \__scontents_esphack:
811    {
812      \scan_stop:
813      \mode_if_horizontal:T
```

```
814        {
815          \int_set_eq:NN \tex_spacefactor:D \l__scontents_save_sf_int
816          \dim_compare:nNnT { \l__scontents_save_skip } > { \c_zero_skip }
817            {
818              \skip_if_eq:nnT { \tex_lastskip:D } { \c_zero_skip }
819                {
820                  \nobreak
821                  \skip_horizontal:n { \c_zero_skip }
822                }
823              \tex_ignorespaces:D
824            }
825        }
826      }
827  ⟨/core⟩
828  ⟨∗latex⟩
829  \cs_gset_eq:NN \__scontents_bsphack: \@bsphack
830  \cs_gset_eq:NN \__scontents_esphack: \@esphack
831  ⟨/latex⟩
```

(*End of definition for* __scontents_bsphack: *and* __scontents_esphack:*.*)

\Scontents

__scontents_Scontents_internal:nn

__scontents_norm_arg:n

__scontents_verb_arg:w

The \Scontents command starts by parsing an optional argument to the function __scontents_-Scontents_internal:nn then delegates to __scontents_verb_arg:w or __scontents_norm_arg:n depending whether a star (∗) argument is present.

```
832  ⟨∗loader⟩
833  \NewDocumentCommand \Scontents { !s !O{} }
834    { \__scontents_Scontents_internal:nn {#1} {#2} }
835  ⟨/loader⟩
836  ⟨∗core⟩
837  \cs_new_protected:Npn \__scontents_Scontents_internal:nn #1 #2
838    {
839      \__scontents_bsphack:
840      \group_begin:
841        \tl_if_novalue:nF {#2}
842          { \keys_set:nn { scontents / Scontents } {#2} }
843        \char_set_catcode_active:n { 9 }
844        \bool_if:NTF #1
845          { \__scontents_verb_arg:w }
846          { \__scontents_norm_arg:n }
847    }
```

The function __scontents_norm_arg:n grabs a normal argument, adds it to the seq variable and optionally prints it.

```
848  \cs_new_protected:Npn \__scontents_norm_arg:n #1
849    {
850      \tl_set:Nn \l__scontents_temp_tl {#1}
851      \__scontents_Scontents_finish:
852    }
```

The function __scontents_verb_arg:w grabs a verbatim argument using xparse's +v argument parser.

```
853  ⟨/core⟩
854  ⟨∗loader⟩
855  \NewDocumentCommand \__scontents_verb_arg:w { +v }
856    { \__scontents_verb_arg_internal:n {#1} }
857  ⟨/loader⟩
858  ⟨∗core⟩
```

(*End of definition for* \Scontents *and others. This function is documented on page 5.*)

__scontents_verb_arg_internal:n

__scontents_Scontents_finish:

__scontents_file_write_cmd:nn

__scontents_file_write_cmd:VV

The function __scontents_verb_arg_internal:n replace all \^^M and \obeyedline added by +v argument added in LaTeX release 2024-06-01 by \^^J then adds it to the seq variable.

```
859  \cs_new_protected:Npn \__scontents_verb_arg_internal:n #1
860    {
861      \tl_set:Nn \l__scontents_temp_tl {#1}
862      \cs_if_exist:NT \obeyedline
863        {
864          \tl_replace_all:Nxx \l__scontents_temp_tl { \obeyedline } { \iow_char:N \^^M }
865        }
```

```
866          \tl_replace_all:Nxx \l__scontents_temp_tl { \iow_char:N \^^M } { \iow_char:N \^^J }
867          \__scontents_Scontents_finish:
868     }
869 \cs_new_protected:Npn \__scontents_Scontents_finish:
870     {
871         \__scontents_file_write_cmd:VV \l__scontents_fname_out_tl \l__scontents_temp_tl
872         \__scontents_finish_storing:NNN
873           \l__scontents_temp_tl
874           \l__scontents_name_seq_cmd_tl
875           \l__scontents_print_cmd_bool
876         \use:x
877           {
878       \group_end:
879       \bool_if:NF \l__scontents_print_cmd_bool { \__scontents_esphack: }
880           }
881     }
882 \cs_new_protected:Npn \__scontents_file_write_cmd:nn #1#2
883     {
884       \__scontents_file_if_writable:nT {#1}
885         {
886           \iow_open:Nn \l__scontents_file_iow {#1}
887           \iow_now:Nn  \l__scontents_file_iow {#2}
888           \iow_close:N \l__scontents_file_iow
889         }
890     }
891 \cs_generate_variant:Nn \__scontents_file_write_cmd:nn { VV }
892 \prg_new_protected_conditional:Npnn \__scontents_file_if_writable:n #1 { T, F, TF }
893     {
894       \bool_if:NTF \l__scontents_writing_bool
895         {
896           \file_if_exist:nTF {#1}
897             {
898               \bool_if:NTF \l__scontents_overwrite_bool
899                 {
900                   \msg_warning:nnx { scontents } { overwrite-file } {#1}
901                   \prg_return_true:
902                 }
903                 {
904                   \msg_warning:nnx { scontents } { not-writing } {#1}
905                   \prg_return_false:
906                 }
907             }
908             {
909               \msg_warning:nnx { scontents } { writing-file } {#1}
910               \prg_return_true:
911             }
912         }
913       { \prg_return_false: }
914     }
```

(*End of definition for* `\__scontents_verb_arg_internal:n`, `\__scontents_Scontents_finish:`, *and* `\__scontents_file_-write_cmd:nn`.)

12.13   The command `\getstored`

`\getstored`

`\__scontents_getstored_internal:nn`

User command `\getstored` to extract ⟨*stored content*⟩ in seq (robust).

```
915 ⟨/core⟩
916 ⟨∗loader⟩
917 \NewDocumentCommand \getstored { O{-1} m }
918   { \__scontents_getstored_internal:nn {#1} {#2} }
919 ⟨/loader⟩
920 ⟨∗core⟩
921 \cs_new_protected:Npn \__scontents_getstored_internal:nn #1 #2
922   {
923       \group_begin:
924         \int_set:Nn \tex_newlinechar:D { `\^^J }
925         \__scontents_rescan_tokens:x
926           {
927       \endgroup % This assumes \catcode`\\=0... Things might go off otherwise.
928       \__scontents_getfrom_seq:nn {#1} {#2}
929           }
```

```
930      }
```

(*End of definition for* `\getstored` *and* `\__scontents_getstored_internal:nn`*. This function is documented on page* 6*.*)

12.14   The command `\foreachsc`

\foreachsc

__scontents_foreachsc_internal:nn

__scontents_foreach_add_body:n

User command `\foreachsc` to loop over ⟨*stored content*⟩ in seq.

```
931  ⟨/core⟩
932  ⟨∗loader⟩
933  \NewDocumentCommand \foreachsc { o m }
934    { \__scontents_foreachsc_internal:nn {#1} {#2} }
935  ⟨/loader⟩
936  ⟨∗core⟩
937  \cs_new_protected:Npn \__scontents_foreachsc_internal:nn #1 #2
938    {
939      \group_begin:
940        \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / foreachsc } {#1} }
941        \tl_set:Nn \l__scontents_foreach_name_seq_tl {#2}
942        \seq_clear:N \l__scontents_foreach_print_seq
943        \bool_if:NF \l__scontents_foreach_stop_bool
944          {
945            \int_set:Nn \l__scontents_foreach_stop_int
946              { \seq_count:c { g__scontents_name_#2_seq } }
947          }
948        \int_step_function:nnnN
949          { \l__scontents_foreach_start_int }
950          { \l__scontents_foreach_step_int }
951          { \l__scontents_foreach_stop_int }
952          \__scontents_foreach_add_body:n
953        \tl_gset:Nx \g__scontents_temp_tl
954          {
955            \exp_args:NNV \seq_use:Nn
956              \l__scontents_foreach_print_seq \l__scontents_foreach_sep_tl
957          }
958      \group_end:
959      \exp_after:wN \tl_gclear:N
960      \exp_after:wN \g__scontents_temp_tl
961        \g__scontents_temp_tl
962    }
963  \cs_new_protected:Npn \__scontents_foreach_add_body:n #1
964    {
965      \seq_put_right:Nx \l__scontents_foreach_print_seq
966        {
967          \bool_if:NT \l__scontents_foreach_before_bool
968            { \exp_not:V \l__scontents_foreach_before_tl }
969          \bool_if:NTF \l__scontents_foreach_wrapper_bool
970            { \__scontents_foreach_wrapper:n }
971            { \use:n }
972            { \getstored [#1] { \tl_use:N \l__scontents_foreach_name_seq_tl } }
973          \bool_if:NT \l__scontents_foreach_after_bool
974            { \exp_not:V \l__scontents_foreach_after_tl }
975        }
976    }
```

(*End of definition for* `\foreachsc`*,* `\__scontents_foreachsc_internal:nn`*, and* `\__scontents_foreach_add_body:n`*. This function is documented on page* 6*.*)

12.15   The command `\typestored`

\typestored

__scontents_typestored_internal:nn

__scontents_verb_print:N

__scontents_xverb:w

The `\typestored` commands fetches a buffer from memory, prints it to the log file, and then calls `\__scontents_verb_print:N`.

```
977  ⟨/core⟩
978  ⟨∗loader⟩
979  \NewDocumentCommand \typestored { o m }
980    { \__scontents_typestored_internal:nn {#1} {#2} }
981  ⟨/loader⟩
982  ⟨∗core⟩
983  \cs_new_protected:Npn \__scontents_typestored_internal:nn #1 #2
984    {
985      \__scontents_bsphack:
```

```
986  \group_begin:
987    \seq_clear:N \l__scontents_seq_item_seq
988    \str_set:Nx \l__scontents_cur_seq_name_str {#2}
989    \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / typemeaning } {#1} }
990    \seq_if_empty:NT \l__scontents_seq_item_seq
991      { \seq_set_from_clist:Nn \l__scontents_seq_item_seq { 1 } }
992    \tl_set:Nx \l__scontents_temp_tl
993      { \__scontents_getfrom_seq:Nn \l__scontents_seq_item_seq {#2} }
994    \__scontents_remove_trailing_eol:N \l__scontents_temp_tl
995    \tl_replace_all:Nxn \l__scontents_temp_tl \c__scontents_hidden_space_str { ^^J }
996    \tl_log:N \l__scontents_temp_tl
997    \tl_if_empty:NF \l__scontents_temp_tl
998      {
999        \bool_if:NT \l__scontents_print_aux_bool
1000          {
1001            \__scontents_verb_print:N \l__scontents_temp_tl
1002          }
1003      }
1004    \__scontents_file_write_cmd:VV \l__scontents_fname_out_tl \l__scontents_temp_tl
1005    \use:x
1006      {
1007  \group_end:
1008    \bool_if:NF \l__scontents_print_aux_bool { \__scontents_esphack: }
1009      }
1010  }
```

The `\__scontents_verb_print:N` macro is defined with active carriage return (ASCII 13) characters to mimic an actual verbatim environment "on the loose". The contents of the environment are placed in a `verbatimsc` environment and rescanned using `\__scontents_rescan_tokens:x`.

```
1011  \group_begin:
1012    \char_set_catcode_active:N \^^M
1013    \cs_new_protected:Npn \__scontents_verb_print:N #1
1014      {
1015        \tl_if_blank:VT #1
1016          { \msg_error:nnn { scontents } { empty-variable } {#1} }
1017        \cs_set_eq:NN \__scontents_verb_print_EOL: ^^M
1018        \cs_set_eq:NN ^^M \scan_stop:
1019        \cs_set_eq:cN { do@noligs } \__scontents_do_noligs:N
1020        \int_set:Nn \tex_newlinechar:D { `\^^J }
1021        \__scontents_rescan_tokens:x
1022          {
1023            \__scontents_format_case:nnn
1024              { \exp_not:N \begin{verbatimsc} } % LaTeX
1025              { \verbatimsc } % Plain/Generic
1026              { \startverbatimsc } % ConTeXt
1027              ^^M
1028            \exp_not:V #1 ^^M
1029            \g__scontents_end_verbatimsc_tl
1030          }
1031        \cs_set_eq:NN ^^M \__scontents_verb_print_EOL:
1032      }
1033  \group_end:
1034  \cs_new_protected:Npn \__scontents_xverb:
1035    {
1036      \char_set_catcode_active:n { 9 }
1037      \char_set_active_eq:nN { 9 } \__scontents_tabs_to_spaces:
1038      \__scontents_xverb:w
1039    }
1040  ⟨/core⟩
```

(End of definition for `\typestored` *and others. This function is documented on page* *6*.*)*

12.16   The command `\mergesc`

The `\mergesc` command parses a list given as argument, and just assembles it as a temporary internal sequence, then passes it to the requested command.

`\mergesc`

`\__scontents_mergesc_output_cmd:nn`
`\__scontents_mergesc_internal:nn`
`\__scontents_mergesc_parse_list:n`
`\__scontents_remove_trailing_eol:N`
`\__scontents_remove_trailing_eol:w`
`\__scontents_parse_mergesc:nw`
`\__scontents_parse_mergesc_aux:nw`

```
1041  ⟨*loader⟩
1042  \NewDocumentCommand \mergesc { o m }
1043    { \__scontents_mergesc_internal:nn {#1} {#2} }
1044  ⟨/loader⟩
```

```
1045  ⟨∗core⟩
1046  \keys_define:nn { scontents / mergesc }
1047    {
1048      , typestored .code:n =
1049        { \cs_set_eq:NN \__scontents_mergesc_output_cmd:nn \__scontents_typestored_internal:nn }
1050      , meaningsc .code:n =
1051        { \cs_set_eq:NN \__scontents_mergesc_output_cmd:nn \__scontents_meaningsc_internal:nn }
1052    }
1053  \cs_new_protected:Npn \__scontents_mergesc_output_cmd:nn #1 #2
1054    { \msg_error:nn { scontents } { mergesc-missing-cmd } }
1055  \msg_new:nnn { scontents } { mergesc-missing-cmd }
1056    { Missing~output~command~for~\iow_char:N\\mergesc~\msg_line_context:. }
1057  \cs_new_protected:Npn \__scontents_mergesc_internal:nn #1 #2
1058    {
1059      \group_begin:
1060        \tl_clear:N \l__scontents_keys_tl
1061        \tl_if_novalue:nF {#1}
1062          {
1063            % Add print-cmd here :D
1064            \keys_define:nn { scontents / typemeaning }
1065              {
1066                print-cmd .bool_set:N = \l__scontents_print_aux_bool,
1067                print-cmd .initial:n = false,
1068                print-cmd .default:n = true,
1069              }
1070            \keys_set_known:nnN { scontents / mergesc } {#1} \l__scontents_keys_tl
1071          }
1072        \seq_gclear:c { g__scontents_name_sc!internal_seq }
1073        \__scontents_mergesc_parse_list:n {#2}
1074        \exp_args:Nx \__scontents_mergesc_output_cmd:nn
1075          { 1-end, \exp_not:V \l__scontents_keys_tl } { sc!internal }
1076      \group_end:
1077    }

1078  \cs_new_protected:Npn \__scontents_mergesc_parse_list:n #1
1079    {
1080      \clist_map_inline:nn {#1} { \__scontents_parse_mergesc:nw ##1 \s__scontents_stop }
1081      \seq_gpop_right:cN { g__scontents_name_sc!internal_seq } \l__scontents_temp_tl
1082      \__scontents_remove_trailing_eol:N \l__scontents_temp_tl
1083      \seq_gput_right:cV { g__scontents_name_sc!internal_seq } \l__scontents_temp_tl
1084    }
1085  \cs_new_protected:Npx \__scontents_remove_trailing_eol:N #1
1086    {
1087      \exp_not:N \exp_after:wN \exp_not:N \__scontents_remove_trailing_eol:w
1088        #1 \s__scontents_stop \c__scontents_hidden_space_str \s__scontents_stop \s__scontents_mark #1
1089    }
1090  \use:e
1091    {
1092      \cs_new_protected:Npn \exp_not:N \__scontents_remove_trailing_eol:w #1
1093          \c__scontents_hidden_space_str \s__scontents_stop #2 \s__scontents_mark #3
1094    } {
1095        \tl_set:Nx #3
1096          {
1097            \tl_if_empty:nTF {#2}
1098              { \exp_not:o { \__scontents_use_delimit_by_s_stop:nw #1 } }
1099              { \exp_not:n {#1} }
1100          }
1101      }
1102  \cs_new_protected:Npn \__scontents_parse_mergesc:nw #1
1103    {
1104      \peek_remove_spaces:n
1105        {
1106          \peek_charcode:NTF [ % ]
1107            { \__scontents_parse_mergesc_aux:nw {#1} }
1108            { \__scontents_parse_mergesc_aux:nw {#1} [ 1-\seq_count:c { g__scontents_name_#1_seq } ] ] }
1109        }
1110    }
1111  \cs_new_protected:Npn \__scontents_parse_mergesc_aux:nw #1 [#2]
1112    {
1113      \seq_clear:N \l__scontents_seq_item_seq
1114      \clist_map_inline:nn {#2}
```

```
1115        { \__scontents_parse_mergesc_range:nw {#1} ##1 - \q__scontents_mark - \s__scontents_mark }
1116      \seq_map_inline:Nn \l__scontents_seq_item_seq
1117        {
1118          \seq_gput_right:cx { g__scontents_name_sc!internal_seq }
1119            { \seq_item:cn { g__scontents_name_#1_seq } {##1} }
1120        }
1121      \__scontents_use_none_delimit_by_s_stop:w
1122    }
1123  \cs_new_protected:Npn \__scontents_parse_mergesc_range:nw #1 #2 - #3 - #4 \s__scontents_mark
1124    {
1125      \cs_set_protected:Npn \__scontents_tmp:w ##1
1126        {
1127          \msg_error:nnxxx { scontents } { index-out-of-range }
1128            {##1} {#1} { \seq_count:c { g__scontents_name_#1_seq } }
1129        }
1130      \__scontents_range_parser:nnxn {#2} {#3}
1131        { \seq_count:c { g__scontents_name_#1_seq } }
1132        { \__scontents_tmp:w }
1133    }
1134  ⟨/core⟩
```

(*End of definition for \mergesc and others. This function is documented on page 6.*)

verbatimsc
\startverbatimsc
\stopverbatimsc

Finally the LaTeX and ConTeXt version of `verbatimsc` environment is defined.

The macro \endverbatim in the second argument of the `verbatimsc` environment is only needed for compatibility with the verbatim package.

```
1135  ⟨∗loader⟩
1136  ⟨∗!context⟩
1137  \use:x
1138    {
1139      \cs_new_protected:Npn \exp_not:N \__scontents_xverb:w
1140        ##1 \g__scontents_end_verbatimsc_tl
1141  ⟨latex⟩     { ##1 \exp_not:N \end{verbatimsc} }
1142  ⟨plain⟩     { ##1 \exp_not:N \endverbatimsc }
1143  ⟨context⟩    { ##1 \exp_not:N \stopverbatimsc }
1144    }
1145  ⟨/!context⟩
1146  ⟨∗latex⟩
1147  \NewDocumentEnvironment { verbatimsc } { }
1148    {
1149      \cs_set_eq:cN { @xverbatim } \__scontents_xverb:
1150      \verbatim
1151    }
1152    { \endverbatim }
1153  ⟨/latex⟩
1154  ⟨context⟩\definetyping[verbatimsc]
1155  ⟨/loader⟩
1156  ⟨∗core⟩
```

(*End of definition for verbatimsc , \startverbatimsc , and \stopverbatimsc. These functions are documented on page 7.*)

12.16.1   Some auxiliaries functions

__scontents_tabs_to_spaces:

In a verbatim context the TAB character is made active and set equal to __scontents_tabs_to_spaces:, to produce as many spaces as the `width-tab` key was set to.

```
1157  \cs_new:Npn \__scontents_tabs_to_spaces:
1158    { \prg_replicate:nn { \l__scontents_tab_width_int } { ~ } }
```

(*End of definition for __scontents_tabs_to_spaces:.*)

__scontents_do_noligs:N

__scontents_do_noligs:N is an alternative definition for LaTeX 2_ε's \do@noligs which makes sure to not consume following space tokens. The LaTeX 2_ε version ends with \char`#1, which leaves TeX still looking for an ⟨*optional space*⟩.

```
1159  \cs_new_protected:Npn \__scontents_do_noligs:N #1
1160    {
1161      \char_set_catcode_active:N #1
1162      \cs_set:cpx { __scontents_active_char_ \token_to_str:N #1 : }
1163        {
```

```
1164              \mode_leave_vertical:
1165              \tex_kern:D \c_zero_dim
1166              \tex_char:D `\exp_not:N #1
1167            }
1168          \char_set_active_eq:Nc #1 { __scontents_active_char_ \token_to_str:N #1 : }
1169        }
```

(*End of definition for* `\__scontents_do_noligs:N`.)

`\__scontents_tl_if_head_is_q_mark:nTF`   Tests if the head of the token list is `\q__scontents_mark`.

```
1170  \prg_new_protected_conditional:Npnn \__scontents_tl_if_head_is_q_mark:n #1
1171    { T, F, TF }
1172    {
1173      \exp_after:wN \if_meaning:w
1174        \exp_after:wN \q__scontents_mark \__scontents_use_i_delimit_by_s_stop:nw #1 ? \s__scontents_stop
1175        \prg_return_true:
1176      \else:
1177        \prg_return_false:
1178      \fi:
1179    }
```

(*End of definition for* `\__scontents_tl_if_head_is_q_mark:nTF`.)

`\__scontents_set_active_eq:NN`
`\__scontents_make_control_chars_active:`
`\__scontents_plain_disable_outer_par:`

Shortcut definitions for common catcode changes. The `^^L` needs a special treatment in non-LaTeX mode because in Plain TeX it is an `\outer` token.

```
1180  \cs_new_protected:Npn \__scontents_set_active_eq:NN #1
1181    {
1182      \char_set_catcode_active:N #1
1183      \char_set_active_eq:NN #1
1184    }
1185  ⟨/core⟩
1186  ⟨∗loader⟩
1187  \group_begin:
1188  ⟨plain⟩  \char_set_catcode_active:n { `\∗ }
1189      \cs_new_protected:Npn \__scontents_plain_disable_outer_par:
1190  ⟨∗plain⟩
1191      {
1192        \group_begin:
1193          \char_set_lccode:nn { `\∗ } { `\^^L }
1194          \tex_lowercase:D { \group_end:
1195          \tex_let:D ∗ \scan_stop:
1196        }
1197      }
1198  ⟨/plain⟩
1199  ⟨latex|context⟩      { }
1200  \group_end:
1201  ⟨/loader⟩
1202  ⟨∗core⟩
1203  \group_begin:
1204    \char_set_catcode_active:N \∗
1205    \cs_new_protected:Npn \__scontents_make_control_chars_active:
1206      {
1207        \__scontents_plain_disable_outer_par:
1208        \__scontents_set_active_eq:NN \^^I \__scontents_tab:
1209        \__scontents_set_active_eq:NN \^^L \__scontents_par:
1210        \__scontents_set_active_eq:NN \^^M \__scontents_ret:w
1211      }
1212  \group_end:
```

(*End of definition for* `\__scontents_set_active_eq:NN`, `\__scontents_make_control_chars_active:`, *and* `\__scontents_-plain_disable_outer_par:`.)

12.17 The command `\setupsc`

User command `\setupsc` to setup module.

`\setupsc`   A user-level wrapper for `\keys_set:nn{ scontents }`.

```
1213  ⟨/core⟩
1214  ⟨∗loader⟩
```

```
1215  \NewDocumentCommand \setupsc { +m }
1216    { \keys_set:nn { scontents } {#1} }
1217  ⟨/loader⟩
1218  ⟨*core⟩
```

(*End of definition for* `\setupsc`*. This function is documented on page* 3*.*)

12.18   The command `\meaningsc`

`\meaningsc`
`\__scontents_meaningsc_internal:nn`
`\__scontents_meaningsc:n`

User command `\meaningsc` to see content stored in seq.

```
1219  ⟨/core⟩
1220  ⟨*loader⟩
1221  \NewDocumentCommand \meaningsc { o m }
1222    { \__scontents_meaningsc_internal:nn {#1} {#2} }
1223  ⟨/loader⟩
1224  ⟨*core⟩
1225  \cs_new_protected:Npn \__scontents_meaningsc_internal:nn #1 #2
1226    {
1227      \group_begin:
1228        \seq_clear:N \l__scontents_seq_item_seq
1229        \str_set:Nx \l__scontents_cur_seq_name_str {#2}
1230        \tl_if_novalue:nF {#1} { \keys_set:nn { scontents / typemeaning } {#1} }
1231        \seq_if_empty:NT \l__scontents_seq_item_seq
1232          { \seq_set_from_clist:Nn \l__scontents_seq_item_seq { 1 } }
1233        \__scontents_meaningsc:n {#2}
1234      \group_end:
1235    }
1236  \group_begin:
1237    \char_set_catcode_active:N \^^I
1238    \cs_new_protected:Npn \__scontents_meaningsc:n #1
1239      {
1240        \tl_set:Nx \l__scontents_temp_tl
1241          { \__scontents_getfrom_seq:Nn \l__scontents_seq_item_seq {#1} }
1242        \tl_replace_all:Nxn \l__scontents_temp_tl { \iow_char:N \^^J } { ~ }
1243        \tl_replace_all:Nxn \l__scontents_temp_tl \c__scontents_hidden_space_str { ~ }
1244        \tl_log:N \l__scontents_temp_tl
1245        \tl_use:N \l__scontents_verb_font_tl
1246        \tl_replace_all:Nnx \l__scontents_temp_tl { ^^I } { \__scontents_tabs_to_spaces: }
1247        \cs_replacement_spec:N \l__scontents_temp_tl
1248      }
1249  \group_end:
```

(*End of definition for* `\meaningsc`*,* `\__scontents_meaningsc_internal:nn`*, and* `\__scontents_meaningsc:n`*. This function is documented on page* 7*.*)

12.19   The command `\countsc`

`\countsc`

User command `\countsc` to count number of contents stored in seq.

```
1250  ⟨/core⟩
1251  ⟨*loader⟩
1252  \NewExpandableDocumentCommand \countsc { m }
1253    { \seq_count:c { g__scontents_name_#1_seq } }
1254  ⟨/loader⟩
1255  ⟨*core⟩
```

(*End of definition for* `\countsc`*. This function is documented on page* 7*.*)

12.20   The command `\cleanseqsc`

`\cleanseqsc`

A user command `\cleanseqsc` to clear (remove) a defined seq.

```
1256  ⟨/core⟩
1257  ⟨*loader⟩
1258  \NewDocumentCommand \cleanseqsc { m }
1259    { \seq_gclear_new:c { g__scontents_name_#1_seq } }
1260  ⟨/loader⟩
1261  ⟨*core⟩
```

(*End of definition for* `\cleanseqsc`*. This function is documented on page* 7*.*)

12.21   Warning and error messages

Warning and error messages used throughout the package.

```
1262 \msg_new:nnn { scontents } { junk-after-begin }
1263   {
1264     Junk~characters~#1~\msg_line_context: :
1265     \\ \\
1266     #2
1267   }
1268 \msg_new:nnnn { scontents } { env-already-defined }
1269   { Environment~'#1'~already~defined! }
1270   {
1271     You~have~used~\newenvsc
1272     with~an~environment~that~already~has~a~definition. \\ \\
1273     The~existing~definition~of~'#1'~will~not~be~altered.
1274   }
1275 \msg_new:nnn { scontents } { empty-stored-content }
1276   { Empty~value~for~key~'getstored'~\msg_line_context:. }
1277 \msg_new:nnn { scontents } { empty-variable }
1278   { Variable~'#1'~empty~\msg_line_context:. }
1279 \msg_new:nnn { scontents } { overwrite-file }
1280   { Overwriting~file~'#1'. }
1281 \msg_new:nnn { scontents } { writing-file }
1282   { Writing~file~'#1'. }
1283 \msg_new:nnn { scontents } { not-writing }
1284   { File~`#1'~already~exists.~Not~writing. }
1285 \msg_new:nnn { scontents } { rescanning-text }
1286   { Rescanning~text~'#1'~after~\c_backslash_str end{#2}~\msg_line_context:.}
1287 \msg_new:nnn { scontents } { multiple-begin }
1288   { Multiple~\c_backslash_str begin{ \l__scontents_env_name_tl }~\msg_line_context:.}
1289 \msg_new:nnn { scontents } { undefined-storage }
1290   { Storage~named~'#1'~is~not~defined. }
1291 \msg_new:nnn { scontents } { index-out-of-range }
1292   {
1293     \int_compare:nNnTF {#1} = { 0 }
1294       { Index~of~sequence~cannot~be~zero. }
1295       {
1296         Index~'#1'~out~of~range~for~'#2'.~
1297         \int_compare:nNnTF {#1} > { 0 }
1298           { Max = } { Min = -} #3.
1299       }
1300   }
1301 \msg_new:nnnn { scontents } { env-key-unknown }
1302   {
1303     The~key~'#1'~is~unknown~by~environment~
1304     '\l__scontents_env_name_tl'~and~is~being~ignored.
1305   }
1306   {
1307     The~environment~'\l__scontents_env_name_tl'~does~not~have~a~key~called~'#1'.\\
1308     Check~that~you~have~spelled~the~key~name~correctly.
1309   }
1310 \msg_new:nnnn { scontents } { env-key-value-unknown }
1311   {
1312     The~key~'#1=#2'~is~unknown~by~environment~
1313     '\l__scontents_env_name_tl'~and~is~being~ignored.
1314   }
1315   {
1316     The~environment~'\l__scontents_env_name_tl'~does~not~have~a~key~called~'#1'.\\
1317     Check~that~you~have~spelled~the~key~name~correctly.
1318   }
1319 \msg_new:nnnn { scontents } { cmd-key-unknown }
1320   { The~key~'#1'~is~unknown~by~'\c_backslash_str Scontents'~and~is~being~ignored.}
1321   {
1322     The~command~'\c_backslash_str Scontents'~does~not~have~a~key~called~'#1'.\\
1323     Check~that~you~have~spelled~the~key~name~correctly.
1324   }
1325 \msg_new:nnnn { scontents } { cmd-key-value-unknown }
1326   { The~key~'#1=#2'~is~unknown~by~'\c_backslash_str Scontents'~and~is~being~ignored. }
1327   {
1328     The~command~'\c_backslash_str Scontents'~does~not~have~a~key~called~'#1'.\\
1329     Check~that~you~have~spelled~the~key~name~correctly.
```

```
1330       }
1331  \msg_new:nnnn { scontents } { for-key-unknown }
1332    { The~key~'#1'~is~unknown~by~'\c_backslash_str foreachsc'~and~is~being~ignored.}
1333    {
1334      The~command~'\c_backslash_str foreachsc'~does~not~have~a~key~called~'#1'.\\
1335      Check~that~you~have~spelled~the~key~name~correctly.
1336    }
1337  \msg_new:nnnn { scontents } { for-key-value-unknown }
1338    { The~key~'#1=#2'~is~unknown~by~'\c_backslash_str foreachsc'~and~is~being~ignored. }
1339    {
1340      The~command~'\c_backslash_str foreachsc'~does~not~have~a~key~called~'#1'.\\
1341      Check~that~you~have~spelled~the~key~name~correctly.
1342    }
1343  \msg_new:nnnn { scontents } { type-key-unknown }
1344    { The~key~'#1'~is~unknown~and~is~being~ignored. }
1345    {
1346      This~command~does~not~have~a~key~called~'#1'.\\
1347      This~command~only~accepts~the~key~'width-tab'.
1348    }
1349  \msg_new:nnnn { scontents } { type-key-value-unknown }
1350    { The~key~'#1'~to~which~you~passed~'#2'~is~unknown~and~is~being~ignored. }
1351    {
1352      This~command~does~not~have~a~key~called~'#1'.\\
1353      This~command~only~accepts~the~key~'width-tab'.
1354    }
1355  \msg_new:nnn { scontents } { empty-environment }
1356    { environment~'#1'~empty~\msg_line_context:. }
1357  \msg_new:nnnn { scontents } { verbatim-newline }
1358    { Verbatim~argument~of~#1~ended~by~end~of~line. }
1359    {
1360      The~verbatim~argument~of~the~#1~cannot~contain~more~than~one~line,~
1361      but~the~end~
1362      of~the~current~line~has~been~reached.~You~may~have~forgotten~the~
1363      closing~delimiter.
1364      \\ \\
1365      LaTeX~will~ignore~'#2'.
1366    }
1367  \msg_new:nnnn { scontents } { verbatim-tokenized }
1368    { The~verbatim~#1~cannot~be~used~inside~an~argument. }
1369    {
1370      The~#1~takes~a~verbatim~argument.~
1371      It~may~not~appear~within~the~argument~of~another~function.~
1372      It~received~an~illegal~token \tl_if_empty:nF {#3} { ~'#3' } .
1373      \\ \\
1374      LaTeX~will~ignore~'#2'.
1375    }
```

12.22   Finish package

Finish package implementation.

```
1376  ⟨/core⟩
1377  ⟨plain|context⟩\ExplSyntaxOff
```

13 Index of Implementation

The italic numbers denote the pages where the corresponding entry is described, the numbers underlined and all others indicate the line on which they are implemented in the package code.